ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОБІЛЬНИХ ПРИСТРОЇВ.

Частина 2

Електронні методичні вказівки до лабораторних робіт студентам факультету математики, фізики та інформаційних технологій першого (бакалаврського) рівня освіти, галузі 12 «Інформаційні технології»

ОДЕСА
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І. І. МЕЧНИКОВА
ФАКУЛЬТЕТ МАТЕМАТИКИ, ФІЗИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОБІЛЬНИХ ПРИСТРОЇВ

Частина 2

Електронні методичні вказівки до лабораторних робіт
студентам
факультету математики, фізики та інформаційних технологій
першого (бакалаврського) рівня освіти,
галузі 12 «Інформаційні технології»

ОДЕСА
ОНУ
2024

УДК 37.091.33:004.4
Г15

Укладачі:
Ю. О. Гунченко, доктор технічних наук, професор, професор кафедри комп'ютерних систем та технологій;
А. В. Камєнєва, кандидат технічних наук, доцент, доцент кафедри комп'ютерних систем та технологій;
О. М. Зуй, викладач кафедри комп'ютерних систем та технологій;
Є. О. Зудіхін, студент бакалаврату комп'ютерних наук Віденського технічного університету прикладних наук, асистент викладача.

Рецензенти:
Микола Малаксіано - доктор технічних наук, професор, зав. кафедри технічної кібернетики й інформаційних технологій ім. проф. Р.В.Меркта, Одеський національний морський університет;
Вичужанін Володимир Вікторович - доктор технічних наук, професор, завідувач кафедри інформаційних технологій. Національний університет «Одеська політехніка».

Рекомендовано Вченою радою факультету МФІТ
Одеського національного університету імені І. І. Мечникова
Протокол № 9 від 27 червня 2024 р.

Програмне забезпечення мобільних пристроїв. Частина 2. [Електронний ресурс]: метод. вказівки до лаб. робіт студ. факультету математики, фізики та інформаційних технологій першого (бакалавр.) рівня освіти, галузі 12
Г15 «Інформаційні технології» : / уклад.: Ю. О. Гунченко, А. В. Камєнєва, О. М. Зуй, Є. О. Зудіхін. – Одеса: Одес. нац. ун-т ім. І. І. Мечникова, 2024. – 57 с.– 1,23 МБ

Методичні вказівки призначені для виконання лабораторних робіт з дисципліни «Програмне забезпечення мобільних пристроїв» для здобувачів першого (бакалаврського) рівня вищої освіти, галузі знань 12 «Інформаційні технології» факультету математики, фізики та інформаційних технологій Одеського національного університету імені І. І. Мечникова.

УДК 37.091.33:004.4

ЗМІСТ

ВСТУП

Дані електронні методичні вказівки до лабораторних занять підготовлені відповідно до програми курсу «Програмне забезпечення мобільних пристроїв». Навчальна дисципліна відноситься до дисциплін вільного вибору студентів 3 курсу галузі 12 «Інформаційні технології».

Предметом вивчення навчальної дисципліни «Програмне забезпечення мобільних пристроїв» є розгляд аспектів, пов'язаних із створенням, впровадженням та управлінням програмним забезпеченням для мобільних пристроїв.

Метою викладання цієї дисципліни є надання студентам знань та практичних навичок у галузі розробки програмного забезпечення мобільних пристроїв.

Завдання дисципліни «Програмне забезпечення мобільних пристроїв» - надати студентам знання в сфері архітектури та особливостей мобільних пристроїв, специфіки розробки програм для мобільних платформ, основних принципів створення та публікації мобільних додатків.

Лабораторні роботи з дисципліни виконуються з метою закріплення та поглиблення теоретичних та практичних знань та вмінь, набутих у процесі засвоєння всього навчального матеріалу дисципліни. Кожна лабораторна робота містить теоретичні відомості, приклади, контрольні запитання та завдання для самостійної роботи студентів.

Метою даних методичних вказівок є закріплення лекційного матеріалу та вироблення у студентів навичок розробки програмного забезпечення мобільних пристроїв з використанням сучасних інструментів та технологій програмування, навичок проектування інтерфейсу користувача та оптимізації продуктивності мобільних додатків, а також навичок тестування та налагодження мобільних додатків, що підвищують їх надійність та безпеку.

У даних методичних вказівках містяться лабораторні роботи для другої частини курсу «Програмне забезпечення мобільних пристроїв».

ПОРЯДОК ВИКОНАННЯ РОБІТ ТА ОФОРМЛЕННЯ ЗВІТУ

Лабораторні роботи містять теоретичні відомості, розібрані приклади, контрольні запитання та завдання для самостійної роботи. Вони призначені для закріплення та поглиблення теоретичних та практичних знань та вмінь, набутих у процесі засвоєння всього навчального матеріалу дисципліни. Кожна лабораторна робота виконується як мінімум одну пару (2 академічні години).

Перед виконанням самостійного завдання студенти мають ознайомитись з теоретичним матеріалом та відповісти на контрольні запитання.

При оформленні звіту з лабораторної роботи в нього обов’язково треба включати номер і назву роботи, її мету, всі завдання, передбачені у ході цієї роботи. Результати вирішення завдань необхідно наводити в повній мірі, ілюструючи достатньою кількістю різних варіантів вхідних даних та результатів роботи програм. Там, де це передбачено завданням, потрібно зробити порівняння. В кінці звіту з кожної лабораторної роботи зробити висновки.

ЛАБОРАТОРНА РОБОТА №6
ВИКОРИСТАННЯ КОНТЕЙНЕРУ TABLEVIEW ТА ЕЛЕМЕНТУ IMAGE У XAMARIN.

Мета: набуття практичних навичок роботи з контейнером TableView та елементами зображень у Xamarin, вивчення способів інтеграції та керування зображеннями через елемент Image, а також розуміння основних принципів відображення динамічного контенту в мобільних додатках.

Теоретичні відомості

TableView дозволяє створювати таблиці з різним вмістом, причому в таблицях можуть розміщуватися також інші елементи управління. TableView може бути корисним для відображення списку різних налаштувань, виведення даних у вигляді форми, або для рядкового відображення даних.

TableView являє собою список, який може бути налаштований для відображення різних типів даних, включаючи текст, зображення та елементи користувача. Кожен елемент списку може містити один або кілька осередків, які можуть містити різні типи вмісту.

Елементи у TableView організовані у секції (елементи TableSection). Кореневим елементом у TableView є елемент TableRoot, який інкапсулює в собі всі секції. Щоб визначити вміст TableView, треба його властивості Root присвоїти певний об'єкт TableRoot. TableRoot зберігає секції таблиці як об'єктів TableSection. Кожна ж секція, у свою чергу, містить набір окремих осередків або елементів EntryCell.

Приклад 6.1. Використання контейнеру TableView.

```
public MainPage()
      {
          InitializeComponent();
          this.Content = new TableView
          {
              Intent = TableIntent.Form,
              Root = new TableRoot("Data input")
           {
              new TableSection ("Person data")
              {
                  new EntryCell
                  {
                      Label = "Login:",
                      Placeholder = "Input login",
                      Keyboard = Keyboard.Default
                  },
                  new SwitchCell { Text = "Save"}
              },
```

```
                new TableSection ("Contacts")
                {
                    new EntryCell
                    {
                        Label = "Phone:",
                        Placeholder = "Input phone",
                        Keyboard = Keyboard.Telephone
                    },
                    new EntryCell
                    {
                        Label = "Email:",
                        Placeholder = "Input email",
                        Keyboard = Keyboard.Email
                    }
                }
            }
        };
    }
```

Властивість Intent об'єкта TableView допомагає покращити візуальне відображення та сприйняття таблиці користувачем, натякаючи, що елементи всередині таблиці призначені для збору або показу інформації, як у формах. Вона може впливати на стиль відображення таблиці, наприклад, на відступи, групування секцій та візуальне розділення елементів форми.

Приклад 6.2. Використання контейнеру TableView у XAML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppPprobaDel.MainPage">
    <TableView>
        <TableView.Root>
            <TableRoot>
                <TableSection Title="Person data">
                    <EntryCell Label="Login" Keyboard="Default"
Placeholder="Input login" />
                    <SwitchCell Text="Save" />
                </TableSection>
                <TableSection Title="Contacts">
                    <EntryCell Label=" Phone:" Keyboard="Telephone"
Placeholder="Input phone" />
                    <EntryCell Label="Email" Keyboard="Email"
Placeholder="Input email" />
                </TableSection>
            </TableRoot>
        </TableView.Root>
    </TableView>
</ContentPage>
```

Типи осередків

При створенні таблиці TableView можна використовувати різні види осередків:

- EntryCell: представляє мітку з текстовим полем для введення даних;
- SwitchCell: представляє мітку з перемикачем;
- TextCell: дві мітки для виведення тексту;
- ImageCell: аналогічна TextCell із включенням зображення;
- ViewCell: вміст та формат відображення даних осередку визначається розробником.

Кожен тип осередків має власний набір властивостей. Наприклад, для комірки типу SwitchCell можна виділити такі властивості:

- Text: представляє текст комірки;
- On: вказує, перебуває у зазначеному чи ні стані.

Для комірки типу EntryCell можна виділити такі властивості:

- Keyboard: тип клавіатури, яка відображається для введення тексту;
- Label: текстова позначка, яка відображається зліва від поля введення;
- LabelColor: колір тексту;
- Placeholder: текст, який відображається до введення тексту;
- Text: сам введений текст;
- HorizontalTextAlignment: горизонтальне вирівнювання тексту.

Види таблиць

Властивість Intent визначає види таблиць і може приймати наступні значення:

- Data: призначена для простого відображення даних;
- Form: представляє форму для введення даних;
- Menu: використовується для виведення меню;
- Settings: використовується для відображення набору налаштувань.

Властивість Intent використовується для вказівки того, який намір має TableView, що допоможе системі вибрати найкращий спосіб відображення його вмісту.

Обробка подій

Елементи TableView підтримують обробку подій. Наприклад, EntryCell при завершенні введення (коли користувач натискає кнопку "Готово" на клавіатурі) генерує подію Completed.

ViewCell при торканні/натисканні генерує подію OnViewCellTapped.

Приклад 6.3. Обробка події EntryCell та SwitchCell.

Код в XAML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppPprobaDel.MainPage">
    <StackLayout>
        <Label x:Name="loginLbl" FontSize="Large" />
        <Label x:Name="saveLbl" FontSize="Large" />
        <TableView>
            <TableView.Root>
                <TableRoot>
                    <TableSection Title="Person data">
                        <EntryCell x:Name="loginEntry" Label="Login"
Keyboard="Default" Placeholder="Input login"
Completed="EntryCell_Completed" />
                        <SwitchCell x:Name="saveSwitch" Text="Save"
OnChanged="SwitchCell_OnChanged" />
                    </TableSection>
                </TableRoot>
            </TableView.Root>
        </TableView>
    </StackLayout>
</ContentPage>
```

Обробка подій у файлі коду C#:

```
private void EntryCell_Completed(object sender, EventArgs e)
        {
            loginLbl.Text = loginEntry.Text;
        }
private void SwitchCell_OnChanged(object sender, ToggledEventArgs e)
        {
            saveLbl.Text = saveSwitch.On.ToString();
        }
```

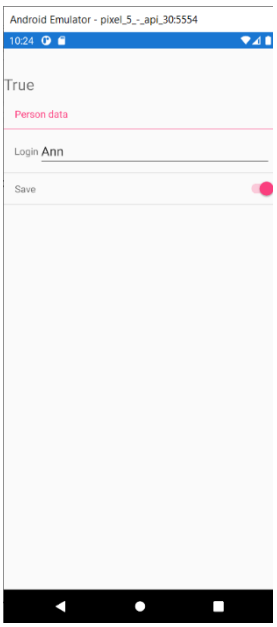

Рис. 6.1. Вигляд інтерфейсу після виконання коду в прикладі 6.3.

Робота із зображеннями. Елемент Image

Вивести зображення на екран можна за допомогою елемента Image.

Локальні зображення

Перед виведенням файл зображення треба додати у проект для Android до папки Resources/Drawable (рис.6.2).

Рис.6.2. Додавання файлу зображення у проект для Android

У вікні Properties треба встановити у цього зображення властивість Build Action: AndroidResource (рис. 6.3).

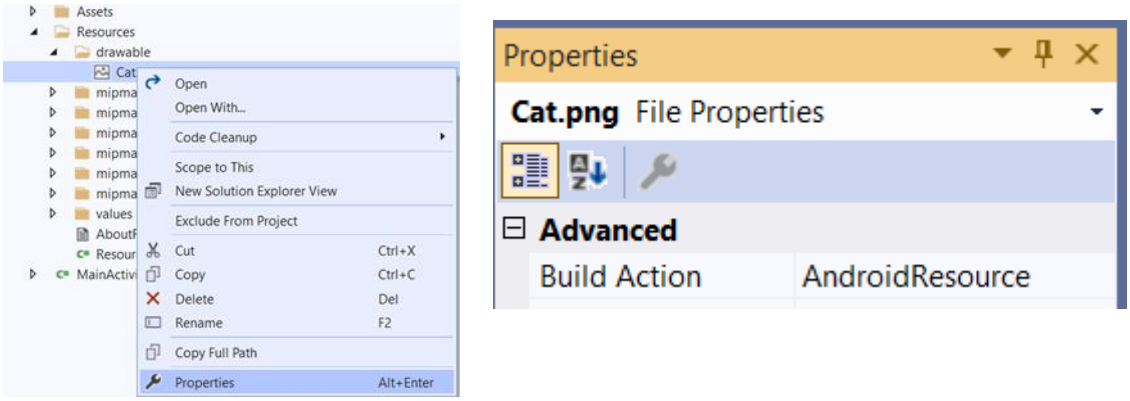

Рис. 6.3. Встановлення властивості файла зображення.

Доступ до зображення

Приклад 6.4. Код виведення зображення в елемент Image на C# в XAML.

```
public MainPage()
      {
          Image image = new Image { Source = "Cat.png" };
          this.Content = image;
      }
```

Аналог в xaml:

```
<Image Source="Cat.png" />
```

Водночас такий підхід може бути незручним. Зберігання всіх зображень локально у проекті може значно збільшити розмір додатка, що негативно впливає на час завантаження та установку додатка на пристрій користувача.

Embedded images

Вбудовані зображення (embedded images) на відміну від вище розглянутих локальних зображень додаються безпосередньо в збірку, що розділяється, в якості ресурсу.

Одна копія збірки, що розділяється, може використовуватися відразу в декількох додатках на одній і тій же машині. Файл зображення треба додати лише у головний проект.

Так, для створення вбудованих зображень потрібно створити в головному проекті новий каталог Images і додати файл зображення (рис. 6.4):

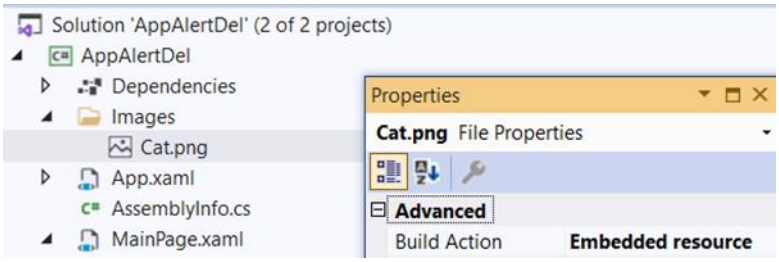

Рис.6.4. Додавання вбудованого зображення у проект для Android

Після додавання зображення до панелі властивостей для поля Build Action потрібно встановити значення Embedded Resources.

Приклад 6.5. Виведення зображення в коді C#.

```
public MainPage()
        {
            InitializeComponent();
            Image image = new Image();
            image.Source =
ImageSource.FromResource("AppAlertDel.Images.Cat.png");
            Content = image;
        }
```

Для отримання ресурсу зображення використовується метод ImageSource.FromResource(). Шлях, який передається, починається з назви проекту, тобто AppAlertDel. Далі йде шлях до зображення усередині проекту. Назва проекту та назва папок на цьому шляху відокремлюються крапками.

Для XAML за умовчанням відсутня можливість транслювати рядковий шлях у потрібний об'єкт. Тому треба вручну писати спеціальне розширення для XAML, яке дозволить транслювати рядковий шлях.

Для цього додається до проекту наступний клас ImageResourceExtension , що реалізує інтерфейс IMarkupExtension:

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

```
 namespace AppAlertDel
{
    [ContentProperty("Source")]
    public class ImageResourceExtension : IMarkupExtension
    {
        public string Source { get; set; }

        public object ProvideValue(IServiceProvider serviceProvider)
        {
            if (Source == null)
            {
                return null;
            }
            var imageSource = ImageSource.FromResource(Source);

            return imageSource;
        }
    }
}
```

IMarkupExtension визначає інтерфейс розширення розмітки XAML Xamarin.Forms. IMarkupExtension — це інтерфейс, який використовується для створення розширень розмітки. Ці розширення дозволяють впроваджувати складну логіку для значень властивостей прямо в XAML.

У методі ProvideValue, який реалізується з інтерфейсу IMarkupExtension, ImageResourceExtension перетворює шлях, використовуючи метод ImageSource.FromResource. Це дозволяє завантажувати зображення з вбудованих ресурсів проекту.

Коли використовується розширення розмітки у XAML, то фактично вказується, що значення властивості має бути отримане не з прямого текстового значення або прив'язки, а з результату виконання коду, наданого розширенням.

Розширення розмітки XAML ImageResourceExtension забезпечує можливість завантаження зображень з ресурсів проекту безпосередньо через XAML, використовуючи спрощений синтаксис.

Приклад 6.6. Застосування класу ImageResourceExtension для завантаження зображення в коді XAML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:AppAlertDel;assembly=AppAlertDel"
             x:Class="AppAlertDel.MainPage">

    <Image Source="{local:ImageResource AppAlertDel.Images.Cat.png}" />

</ContentPage>
```

У XAML-файлі класи та властивості посилаються на XML-елементи та атрибути, а також встановлюються зв'язки між розміткою та кодом.

`xmlns:local="clr-namespace:AppAlertDel;assembly=AppAlertDel"` означає, що використовується CLR-простір імен "AppAlertDel" у програмі. Цей рядок дозволяє вказати цей простір імен за допомогою префікса "local" у XAML-розмітці. Наприклад, цей префікс використовується для посилань на клас з простору імен AppAlertDel:

`<Image Source="{local:ImageResource AppAlertDel.Images.Cat.png}"/>`.

Коли у XAML вказується Source з використанням розширення local:ImageResource, Xamarin.Forms виконує наступні кроки:

- Визначається, що атрибут Source елемента <Image> має оброблятися через розширення розмітки, вказане у local:ImageResource. Це означає, що для отримання значення Source буде використана логіка, визначена у класі ImageResourceExtension.

- ImageResourceExtension отримує рядок Source, який представляє шлях до зображення у ресурсах проекту (наприклад, AppAlertDel.Images.Cat.png).

Завантаження із мережі

Крім локальних картинок Xamarin також підтримує завантаження із мережі.

Приклад 6.7. Завантаження зображення із мережі в коді C# та на XAML.

```
public MainPage()
        {
            InitializeComponent();
            Image image = new Image();
            image.Source = new UriImageSource
            {
                CachingEnabled = false,
                Uri = new
System.Uri("https://sobakino.com/uploads/images/00/00/01/2016/10/21/01ead
1.jpg")
            };
            Content = image;
        }
```

У файлі xaml:

```
<Image Source=
"https://sobakino.com/uploads/images/00/00/01/2016/10/21/01ead1.jpg"/>
```

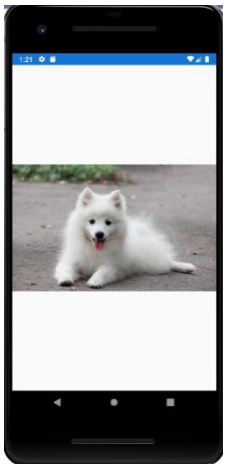

Рис.6.5. Інтерфейс для прикладу 6.7.

Контрольні питання

1. Що таке TableView, і для чого він використовується у Xamarin.Forms?
2. Як організована структура TableView, і які основні елементи входять до його складу?
3. Які типи осередків (cells) можна використовувати у TableView, і які їх характеристики?
4. Як можна обробляти події від різних елементів у TableView, наприклад, від EntryCell або SwitchCell?
5. Які існують способи включення зображень у Xamarin.Forms проекти, і які мають переваги та недоліки локальні та вбудовані зображення?
6. Яким чином можна завантажити зображення із мережі і відобразити його у Xamarin.Forms додатку?

Завдання до лабораторної роботи

1. Розробіть інтерфейс програми (рис. 6.6) для перегляду зображень за допомогою елемента компонування TableView, елементів Image мовою XAML. Image додати в проект окремо для Android, використати вбудовані зображення (embedded images), а також завантажити із мережі.

2. Розробіть інтерфейс за допомогою C#

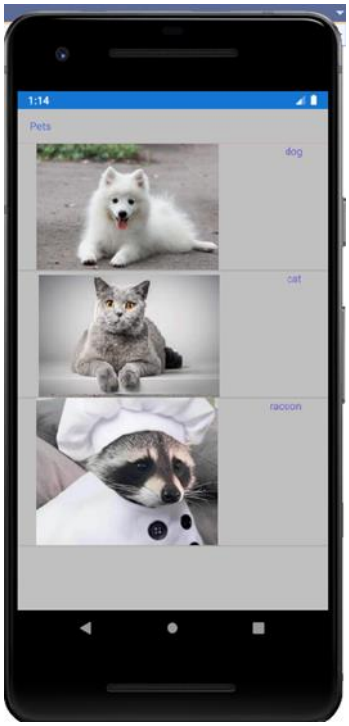

Рис.6.6. Приклад інтерфейсу для лабораторної роботи.

ЛАБОРАТОРНА РОБОТА №7
ВИКОРИСТАННЯ РЕСУРСІВ В ДОДАТКАХ.

Мета: вивчити основи використання ресурсів у мобільних додатках на платформі Xamarin Forms.

Теоретичні відомості

Використання ресурсів в додатках

Для спільного використання одних і тих же компонентів різними елементами Xamarin Forms застосовує концепцію ресурсів. У разі під ресурсами розуміються не допоміжні файли - зображень тощо, які є у додатку, а логічні ресурси, що визначаються в коді C# чи XAML.

Визначення ресурсів

Як ресурс можна визначити будь-який об'єкт. Всі ресурси розміщуються в об'єкті ResourceDictionary. Кожен візуальний об'єкт, наприклад ContentPage або Button, має властивість Resources, яка якраз зберігає об'єкт ResourceDictionary.

Приклад 7.1. Визначення декількох ресурсів.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <Color    x:Key="textColor">#004D40</Color>
            <Color    x:Key="backColor">#80CBC4</Color>
            <x:Double x:Key="fontSize">24</x:Double>
        </ResourceDictionary>
    </ContentPage.Resources>
    <StackLayout>
        <Button Text="UWP"
           TextColor="{StaticResource Key=textColor}"
     BackgroundColor="{StaticResource Key=backColor}"
            FontSize="{StaticResource Key=fontSize}" />
        <Button Text="Android"
           TextColor="{StaticResource Key=textColor}"
     BackgroundColor="{StaticResource Key=backColor}"
            FontSize="{StaticResource Key=fontSize}" />
    </StackLayout>
</ContentPage>
```

Кожен ресурс повинен мати ключ, який задається за допомогою атрибуту x:Key. Це своєрідний унікальний ідентифікатор ресурсу. Наприклад:

```
<Color x:Key="textColor">#004D40</Color>
```

Тут як ресурс визначається об'єкт Color. Цей об'єкт має ключ textColor.

Щоб звернутися до цього ресурсу в коді, потрібно використовувати розширення StaticResource:

```
TextColor="{StaticResource Key=textColor}"
```

Властивість Key через ключ ресурсу посилатиметься на цей ресурс.

При цьому важливо, що на ресурс можуть посилатися кілька елементів. Наприклад, потрібно створити загальний для всіх кнопок фоновий колір. І в цьому випадку простіше визначити колірний ресурс, ніж надавати властивості BackgroundColor у кожної кнопки конкретний колір. А якщо потрібно змінити колір кнопок на інший, то не треба буде змінювати властивість BackgroundColor у всіх кнопок. Досить змінити значення ресурсу.

У той же час, при старті додатку йому потрібен деякий час, щоб знайти потрібні ресурси. Тому використання ресурсів у порівнянні зі стандартними значеннями трохи сповільнює роботу програми.

Рівні ресурсів

Ресурси можуть визначатися на трьох рівнях:

– на рівні окремого елемента керування. Такі ресурси можуть застосовуватися до всіх вкладених елементів, визначених усередині цього елемента;

– на рівні всієї сторінки. Такі ресурси можуть застосовуватись до всіх елементів на сторінці;

– на рівні всього додатка. Ці ресурси доступні з будь-якого місця та з будь-якої сторінки програми.

Вище у прикладі ресурси визначалися лише на рівні сторінки.

Приклад 7.2. Визначення ресурсів на рівні елемента (StackLayout).

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">
    <StackLayout>
        <StackLayout.Resources>
            <ResourceDictionary>
                <Color    x:Key="textColor">#004D40</Color>
                <Color    x:Key="backColor">#80CBC4</Color>
                <x:Double x:Key="fontSize">22</x:Double>
            </ResourceDictionary>
        </StackLayout.Resources>
        <Button Text="UWP"
                  TextColor="{StaticResource Key=textColor}"
            BackgroundColor="{StaticResource Key=backColor}"
                   FontSize="{StaticResource Key=fontSize}" />
```

```
        <Button Text="Android"
                TextColor="{StaticResource Key=textColor}"
          BackgroundColor="{StaticResource Key=backColor}"
                 FontSize="{StaticResource Key=fontSize}" />
    </StackLayout>
</ContentPage>
```

По суті результат у цьому випадку буде той самий, тому що всі елементи все одно визначені всередині StackLayout.

Ресурси програми

Для визначення спільних для всієї програми ресурсів у проекті є файл App.xaml, який пов'язаний з основним файлом програми App.xaml.cs. Для визначення загальних для всієї програми ресурсів файл App.xaml може виглядати так як в прикладі 7.3.

Приклад 7.3. Файл App.xaml.

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.App">

    <Application.Resources>
        <ResourceDictionary>
            <Color    x:Key="textColor">#004D40</Color>
            <Color    x:Key="backColor">#80CBC4</Color>
            <x:Double x:Key="fontSize">22</x:Double>
        </ResourceDictionary>
    </Application.Resources>
</Application>>
```

Тобто у прикладі 7.3 визначені ті самі ресурси, що й раніше, тільки тепер вони будуть доступні для будь-якого елемента на будь-якій сторінці всередині програми. І в цьому випадку можна просто їх використати.

Приклад 7.4. Використання ресурсів програми.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">

<StackLayout>
        <Button Text="UWP"
           TextColor="{StaticResource Key=textColor}"
     BackgroundColor="{StaticResource Key=backColor}"
            FontSize="{StaticResource Key=fontSize}" />
```

```
        <Button Text="Android"
           TextColor="{StaticResource Key=textColor}"
     BackgroundColor="{StaticResource Key=backColor}"
            FontSize="{StaticResource Key=fontSize}" />
    </StackLayout>
</ContentPage>
```

Управління ресурсами у коді C#

Для керування ресурсами застосовуються методи і властивості ResourceDictionary. ResourceDictionary — це спеціалізований словник, який використовується для зберігання та управління ресурсами в XAML-проектах, таких як стилі, шаблони, кольори, строки та інші об'єкти, які можуть бути використані у додатку.

Методи і властивості ResourceDictionary:

– Add(string key, object resource): додає об'єкт із ключем key у словник, причому у словник можна додати будь-який об'єкт, головне йому зіставити ключ,

– Remove(string key): видаляє із словника ресурс із ключем key.

Щоб знайти ресурс у словнику, достатньо звернутися за ключем:
androidButton.TextColor = (Color)Resources["textColor"];

Так як словник ресурсів зберігає об'єкти типу об'єкта, то при отриманні ресурсу його треба привести до потрібного типу.

Приклад 7.5. Використання ресурсів програми у коді C# .

```
using System;
using Xamarin.Forms;
namespace AppResDel
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            Color textColor = Color.FromRgb(0, 77, 64);
            Color backColor = Color.FromRgb(128, 203, 196);
            Double fontSize = 21;

            ResourceDictionary resDict = new ResourceDictionary();
            // adding resources to the dictionary
            resDict.Add("textColor", textColor);
            resDict.Add("backColor", backColor);
            resDict.Add("fontSize",  fontSize);

            // set resource dictionary
            this.Resources = resDict;

            Button uwpButton = new Button { Text = "UWP" };
```

```
            // get resource from dictionary
            uwpButton.TextColor = (Color)Resources["textColor"];
            uwpButton.BackgroundColor =
(Color)Resources["backColor"];
            uwpButton.FontSize = (double)Resources["fontSize"];

            Button androidButton = new Button { Text = "Android" };
            androidButton.TextColor = (Color)Resources["textColor"];
            androidButton.BackgroundColor =
(Color)Resources["backColor"];
            androidButton.FontSize = (double)Resources["fontSize"];

            Content = new StackLayout
            {
                Children = { uwpButton, androidButton }
            };
        }
    }
}
```

Створення багатосторінкового додатку

У Xamarin.Forms багатосторінковий додаток може бути створений за допомогою декількох екземплярів класу ContentPage, кожен з яких представлятиме окрему сторінку в додатку.

Для підтримки навігації на кожній сторінці Page у Xamarin Forms визначено властивість Navigation. Ця властивість представляє інтерфейс INavigation, у якому є метод *Task PushAsync(Page page).* Як параметр передається об'єкт Page - сторінка, на яку треба здійснити перехід.

Також є метод повернення на попередню сторінку *Task<Page> PopAsync().*

Приклад навігації.

Спочатку треба створити новий проект типу Xamarin Forms Portable. Програма буде здійснювати навігацію, тобто переходи між сторінками, тому потрібно додати до проекту сторінки (рис.7.1).

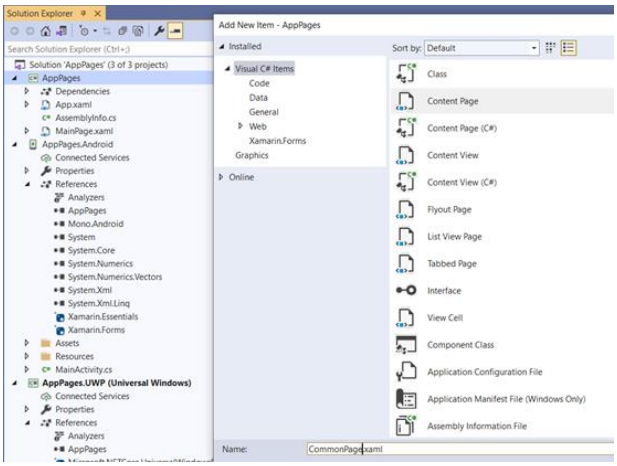

Рис. 7.1. Додавання до проекту нового елементу ContentPage

На доданій сторінці визначається єдина кнопка, натисканням на яку здійснюється перехід назад за допомогою методу Navigation.PopAsync(). Оскільки метод є асинхронним, перед ним ставиться ключове слово await, а сам обробник кнопки позначається за допомогою ключового слова async. Сторінка встановлює заголовок через властивість Title.

Приклад 7.6. Вигляд коду у створеному класі.

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace AppPages
{
    public partial class CommonPage : ContentPage
    {
        public CommonPage()
        {
            Title = "Common Page";
            Button backButton = new Button
            {
                Text = "Back",
                HorizontalOptions = LayoutOptions.Center,
                VerticalOptions = LayoutOptions.Center
            };
            backButton.Clicked += BackButton_Click;
            Content = backButton;
        }
        private async void BackButton_Click(object sender, EventArgs e)
        {
            await Navigation.PopAsync();
        }
    }
}
```

На головну сторінку MainPage додана кнопка, яка здійснює перехід на створену сторінку CommonPage за допомогою методу Navigation.PushAsync().

Приклад 7.7. Вигляд коду для головної сторінки MainPage.

```
using System;
using Xamarin.Forms;
namespace AppPages
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            Title = "Main Page";
            Button toCommonPageBtn = new Button
            {
                Text = "To common page",
                HorizontalOptions = LayoutOptions.Center,
                VerticalOptions = LayoutOptions.CenterAndExpand
            };
            toCommonPageBtn.Clicked += ToCommonPage;

            Content = toCommonPageBtn;
        }
        private async void ToCommonPage(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new CommonPage());
        }
    }
}
```

Щоб з MainPage можна було б переходити до CommonPage, треба як головну сторінку використовувати об'єкт NavigationPage, а не ContentPage, як встановлено за замовчуванням.

Клас NavigationPage в Xamarin є частиною бібліотеки Xamarin.Forms і є контейнерною сторінкою, яка забезпечує навігацію між іншими сторінками програми. NavigationPage містить стек сторінок, на які користувач може переходити вперед і назад, використовуючи функції навігації, такі як PushAsync та PopAsync.

Основне призначення NavigationPage – забезпечити користувачеві можливість переміщатися між сторінками програми. Крім того, він надає ряд вбудованих функцій навігації, таких як відображення кнопки "назад" на верхній панелі програми, зміна заголовка сторінки, а також можливість налаштування стилю верхньої панелі.

Для використання механізму навігації у додатку, необхідно обернути об'єкт MainPage у NavigationPage:

```
MainPage = new NavigationPage(new MainPage());
```

Інакше неможливо використовувати навігацію у додатку. Сам клас NavigationPage візуально тільки додає заголовок сторінки, який встановлюється за допомогою властивості Title і який відображається поряд з іконкою програми.

Приклад 7.8. Змінений код встановлення головної сторінки в класі App.

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace AppPages
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();
            MainPage = new NavigationPage(new MainPage());
        }
        protected override void OnStart()
        {
        }
        protected override void OnSleep()
        {
        }
        protected override void OnResume()
        {
        }
    }
 }
```

Інтерфейс багатосторінкового додатку AppPages наведено на рис. 7.2. Залежно від операційної системи, відображення заголовка може відрізнятися.

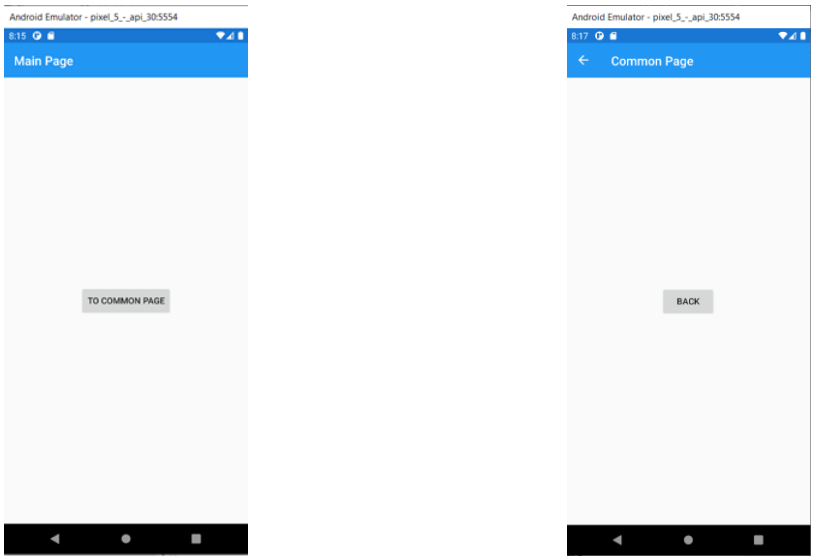

Рис.7.2. Інтерфейс багатосторінкового додатку

Контрольні питання

1. Що таке ResourceDictionary, і яку роль він відіграє у Xamarin Forms?
2. Які типи ресурсів можна визначити у Xamarin Forms і як їх можна використовувати у додатку?
3. Яка різниця між використанням ресурсів на рівні елемента, сторінки та додатка?
4. Як впливає використання ресурсів на продуктивність додатка?
5. Наведіть приклад, як звернутися до ресурсу в коді XAML та C#.

Завдання до лабораторної роботи

Розробіть багатосторінковий додаток (рис. 7.3) для перегляду зображень та інформації про вибраний об’єкт на головній сторінці. Інтерфейси головної та допоміжних сторінок розробити за допомогою елемента компонування RelativeLayout та елементів Image, Button, Label. При натисканні на кнопки DETAILS повинно відкриватися відповідне вікно з більш докладною інформацією про вибраний об’єкт.

Для спільного використання одних і тих же компонентів визначити ресурси на рівні кожної сторінки та на рівні всього додатка.

Предметна область вибирається за бажанням.

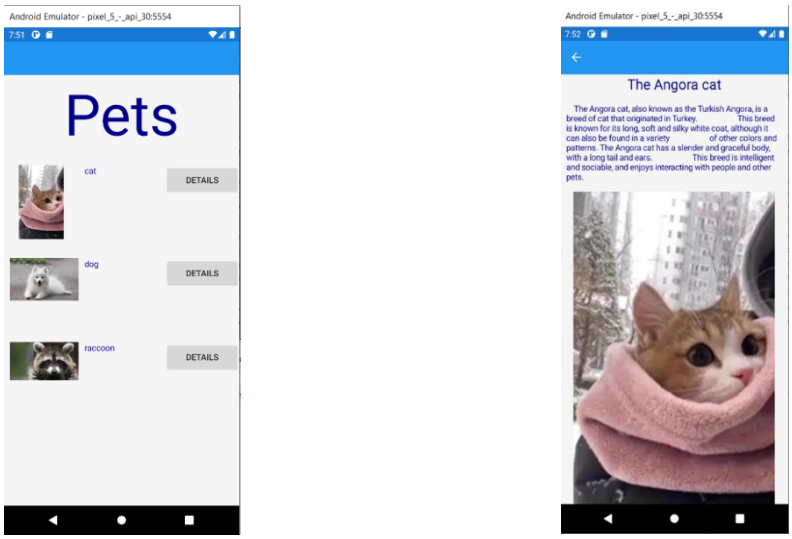

Рис. 7.3. Приклад інтерфейсу багатосторінкового додатку для лабораторної роботи №7

ЛАБОРАТОРНА РОБОТА №8
ВИКОРИСТАННЯ СТИЛІВ В ДОДАТКАХ

Мета роботи: вивчити основи використання стилів у мобільних додатках на платформі Xamarin Forms.

Теоретичні відомості

Визначення стилів

Стилі дозволяють визначити набір деяких властивостей та їх значень, які можуть застосовуватися до елементів. Основне завдання - створити стильову однаковість для елементів інтерфейсу. Стилі зберігаються в ресурсах і відокремлюють значення властивостей елементів від інтерфейсу користувача.

Приклад 8.1. Визначення двох кнопок з однаковим стилем.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">
    <StackLayout>
        <Button Text="UWP" TextColor="#004D40"
BackgroundColor="#80CBC4" FontSize="Large" />
        <Button Text="Android" TextColor="#004D40"
BackgroundColor="#80CBC4" FontSize="Large" />
    </StackLayout>
     </ContentPage>
```

В прикладі 8.1 визначено дві кнопки, які фактично мають той самий стиль: одні й ті ж кольори фону і тексту, а також розмір тексту. Єдина відмінність полягає у тексті кнопки. Однак у цьому випадку потрібно повторюватися і повторно визначати одні й самі властивості й одні й самі значення кожного з елементів.

Приклад 8.2. Визначення двох кнопок з із застосуванням стилів.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Style x:Key="buttonStyle" TargetType="Button">
                <Setter Property="TextColor" Value="#004D40" />
                <Setter Property="BackgroundColor" Value="#80CBC4"
/>
                <Setter Property="FontSize" Value="Large" />
            </Style>
```

```
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout>
        <Button Text="UWP" Style="{StaticResource buttonStyle}" />
        <Button Text="Android" Style="{StaticResource buttonStyle}" />
    </StackLayout>
</ContentPage>
```

Стиль створюється як ресурс за допомогою об'єкта Style, і як будь-який інший ресурс він обов'язково повинен мати ключ. Атрибут TargetType показує, до якого типу належить стиль. У цьому разі це тип Button.

За допомогою колекції Setters визначається група властивостей, що входять до стилю. До неї входять об'єкти Setter, які мають такі властивості:

– Property: вказує на властивість, до якої буде застосовувати цей сетер. При цьому властивість має представляти тип BindableProperty;

– Value: власне значення властивості.

Оскільки стиль визначається як ресурс, для його установки використовуються розширення StaticResource або DynamicResource (якщо стиль динамічний):

```
<Button Text="UWP" Style="{StaticResource buttonStyle}" />
<Button Text="Android" Style="{DynamicResource buttonStyle}" />
```

Також як значення `Value` можна встановлювати посилання на інший ресурс.

Приклад 8.3. Визначення Value для ресурсу за допомогою посилання на інший ресурс.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Color x:Key="greenColor">#004D40</Color>
            <Style x:Key="buttonStyle" TargetType="Button">
                <Setter Property="TextColor" Value="{StaticResource
Key=greenColor}" />
                <Setter Property="BackgroundColor" Value="#80CBC4" />
                <Setter Property="FontSize" Value="Large" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>
    <StackLayout>
        <Button Text="UWP" Style="{StaticResource buttonStyle}" />
        <Button Text="Android" Style="{StaticResource buttonStyle}" />
    </StackLayout>
     </ContentPage>
```

TargetType

Якщо потрібно створити загальний стиль для елементів певного типу, то можна не задавати ключ ресурсу, а достатньо встановити у стилі атрибут TargetType, до якого передається тип елементів.

Приклад 8.4. Створення загального стилю для елементів певного типу.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Color x:Key="greenColor">#004D40</Color>
            <Style TargetType="Button">
                <Setter Property="TextColor" Value="{StaticResource
Key=greenColor}" />
                <Setter Property="BackgroundColor" Value="#80CBC4" />
                <Setter Property="FontSize" Value="Large" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>
    <StackLayout>
        <Button Text="UWP" />
        <Button Text="Android" />
    </StackLayout>
</ContentPage>
```

Тепер у кнопок не треба буде вказувати ресурс стилю, оскільки стиль автоматично застосовуватиметься до всіх об'єктів типу, який вказаний в атрибуті TargetType.

Перевизначення стилів

Стиль дозволяє встановити деякі початкові значення. Однак елемент може перевизначити окремі значення зі стилю.

Приклад 8.5. Перевизначення стилів.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <Style x:Key="buttonStyle" TargetType="Button">
                <Setter Property="TextColor" Value="#004D40" />
                <Setter Property="BackgroundColor" Value="#80CBC4" />
                <Setter Property="FontSize" Value="Large" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>
```

```
    <StackLayout>
        <Button Text="Android" Style="{StaticResource buttonStyle}"
TextColor="Red" />
    </StackLayout>
</ContentPage>
```

В даному випадку кнопка отримує всі значення зі стилю buttonStyle, проте перевизначає колір тексту, так як пряме використання атрибутів елемента має пріоритет над стилем, що приймається.

Встановлення стилів у коді

Для створення стилю в коді використовується об'єкт Style. У конструктор об'єкта Style передається тип, для якого призначений даний стиль - аналогічно використанню атрибуту TargetType в XAML.

Приклад 8.6. Встановлення стилів у коді.

```
public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            Resources = new ResourceDictionary
            {
                {
                    "buttonStyle", new Style(typeof(Button))
                    {
                        Setters =
                        {
                            new Setter
                            {
                                Property = Button.TextColorProperty,
                                Value = Color.FromRgb(0, 77, 64)
                            },
                            new Setter
                            {
                                Property = Button.BackgroundColorProperty,
                                Value = Color.FromRgb(128, 203, 196)
                            },
                            new Setter
                            {
                                Property = Button.FontSizeProperty,
                                Value =
Device.GetNamedSize(NamedSize.Large, typeof(Button))
                            }
                        }
                    }
                }
            };
            Button button1 = new Button { Text = "UWP", Style =
(Style)Resources["buttonStyle"] };
            Button button2 = new Button { Text = "Android", Style =
(Style)Resources["buttonStyle"] };
```

```
            Content = new StackLayout
            {
                Children = { button1, button2 }
            };
        }
}
```

Наслідування стилів

За допомогою властивості BasedOn можна наслідувати один стиль від іншого.

Приклад 8.7. Наслідування стилів.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AppResDel.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Style x:Key="baseButtonStyle" TargetType="Button">
                <Setter Property="FontSize" Value="Large" />
                <Setter Property="FontFamily" Value="Verdana" />
                <Setter Property="TextColor" Value="Red" />
            </Style>
            <Style x:Key="greenButtonStyle" TargetType="Button"
BasedOn="{StaticResource Key=baseButtonStyle}">
                <Setter Property="TextColor" Value="#004D40" />
                <Setter Property="BackgroundColor" Value="#80CBC4" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>
    <StackLayout>
        <Button Text="UWP" Style="{StaticResource Key=greenButtonStyle}"
/>
        <Button Text="Android" Style="{StaticResource
Key=greenButtonStyle}" />
    </StackLayout>
     </ContentPage>
```

Тут у стилю greenButtonStyle атрибут BasedOn вказує на інший стиль baseButtonStyle. І таким чином, стиль greenButtonStyle перейматиме всі установки від baseButtonStyle. У цьому спадковий стиль може перевизначити значення з успадкованого. Часто тут перевизначається значення властивості TextColor.

При наслідуванні стилів важливо, щоб тип елементів, вказаний як значення атрибута TargetType, збігався. Наприклад, у цьому випадку обидва стилі застосовуються до елементів типу Button.

Спадкування в коді здійснюється за допомогою установки у стилю властивості BasedOn:

```
Style basedStyle = new Style(typeof(Button));
Style childStyle = new Style(typeof(Button))
{
    BasedOn = basedStyle
};
```

Контрольні питання

1. Яка мета використання стилів у додатках Xamarin Forms?
2. Опишіть, як визначити стиль у XAML у Xamarin Forms. Які ключові елементи входять у це визначення?
3. Як можна застосувати стиль глобально до всіх контролів певного типу без присвоєння ключа?
4. Для чого використовується атрибут TargetType у ресурсі стилю?
5. Як можна перевизначити конкретні властивості стилю для окремого елемента в Xamarin Forms?
6. Поясніть, як працює успадкування стилів у Xamarin Forms. Що робить властивість BasedOn?

Завдання до лабораторної роботи

Розробіть багатосторінковий додаток (рис.8.1) для перегляду зображень та інформації про вибраний об’єкт на головній сторінці. Інтерфейси головної та допоміжних сторінок розробити за допомогою елемента компонування Grid та елементів Image, Button, Label. При натисканні на кнопки DETAILS повинно відкриватися відповідне вікно з більш докладною інформацією про вибраний об’єкт.

Створити стильову однаковість для елементів інтерфейсу. За допомогою стилів визначити набір властивостей та їх значень, які необхідно застосовувати до використовуваних елементів інтерфейсу.

Визначити та використати стилі у коді на C# та на XAML.

Предметна область вибирається за бажанням.

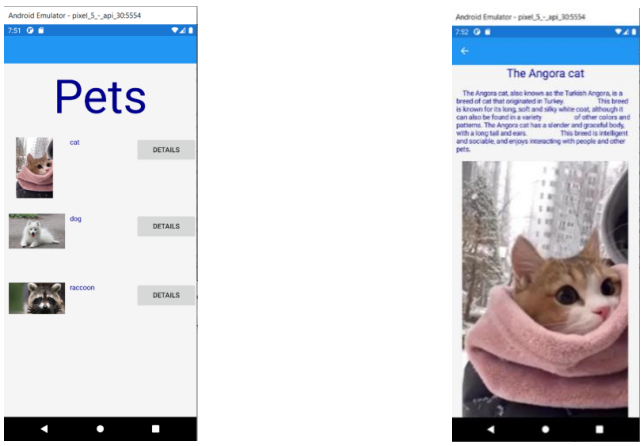

Рис. 8.1. Приклад інтерфейсу багатосторінкового додатку для лабораторної роботи №8

ЛАБОРАТОРНА РОБОТА №9
ЗМІНЮВАННЯ ВЛАСТИВОСТЕЙ ОБ'ЄКТА ЗА ДОПОМОГОЮ ТРИГЕРІВ

Мета роботи: вивчити основи використання тригерів у мобільних додатках на платформі Xamarin Forms.

Теоретичні відомості

Тригери в Xamarin.Forms дозволяють автоматично змінювати властивості одного або кількох елементів у відповідь на зміни в інших властивостях або події.

Тригери в Xamarin — це спеціальні компоненти, які дозволяють змінювати властивості об'єкта або виконувати дії відповідно до змін у його стані без написання додаткового коду управління. Вони часто використовуються в XAML для визначення реакцій на зміни у властивостях об'єктів або стани елементів інтерфейсу.

Існує кілька типів тригерів в Xamarin.Forms:

- Property Triggers: активуються коли властивість об'єкта досягає певного значення. Наприклад, можна змінити колір тексту, коли певне поле вводу стає активним.

- Data Triggers: схожі на property triggers, але активуються відповідно до змін в даних. Вони дозволяють виконувати зміни на основі значень даних, не вдаючись до коду C#.

- Event Triggers: реагують на події, які відбуваються на компоненті. Наприклад, вони можуть викликати анімацію як відповідь на натискання кнопки.

- Multi Triggers: дозволяють задати декілька умов, які всі мають бути виконані для активації тригера.

Тригери забезпечують потужний спосіб для реакції на зміни властивостей або станів елементів UI, дозволяючи дизайнерам і розробникам створювати більш інтерактивні та реактивні інтерфейси без зайвого коду.

Тригери властивостей

Прості тригери властивостей визначаються як елементи стилю з допомогою об'єкта Trigger. Вони стежать за значеннями властивостей. У разі зміни властивостей встановлюють значення інших властивостей за допомогою об'єкта Setter.

Стилі дозволяють визначити набір деяких властивостей та їх значень, які можуть застосовуватися до елементів. Основне завдання - створити стильову однаковість для елементів інтерфейсу. Стилі зберігаються в ресурсах і відокремлюють значення властивостей елементів від інтерфейсу користувача.

Стиль створюється як ресурс за допомогою об'єкта Style, і як будь-який

інший ресурс він обов'язково повинен мати ключ. Атрибут TargetType показує, до якого типу належить стиль.

Стиль визначається елементом Style.Triggers.

У кожного тригера встановлюються три властивості:

- Property: властивість, зміну якого має відстежувати тригер;
- Value: значення властивості, у якому має спрацьовувати тригер;
- TargetType: тип об'єктів, до яких застосовується тригер.

За допомогою колекції Setters визначається група властивостей, що входять до стилю.

Об'єкт Setter у Xamarin використовується для задання значення властивості елементу управління в рамках стилів, тригерів, або шаблонів візуальних станів. Він є ключовою частиною в оформленні і керуванні поведінкою елементів інтерфейсу.

Основні характеристики Setter:

- Property: властивість, яку потрібно змінити. Це може бути будь-яка властивість елементу управління, наприклад BackgroundColor, TextColor, FontSize тощо.

- Value: значення, яке буде присвоєне цій властивості. Значення може бути прямо вказане, або прив'язане до даних чи ресурсів.

Setter може бути частиною колекції Setters у стилі, який визначає зовнішній вигляд та поведінку одного або декількох елементів інтерфейсу.

Оскільки стиль визначається як ресурс, для його установки використовуються розширення StaticResource або DynamicResource (якщо стиль динамічний).

Приклад 9.1. Тригер, що змінює колір шрифту текстового поля Entry при переході до нього.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="lec7del.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Style x:Key="entryStyle" TargetType="Entry">
                <Style.Triggers>
                    <Trigger Property="Entry.IsFocused" Value="True"
TargetType="Entry">
                        <Setter Property="TextColor" Value="Red" />
                    </Trigger>
                </Style.Triggers>
            </Style>
        </ResourceDictionary>
```

```
    </ContentPage.Resources>
    <StackLayout>
        <Entry FontSize="Large" Style="{StaticResource
Key=entryStyle}" />
    </StackLayout>
</ContentPage>
```

Тригер в даному випадку буде спрацьовувати, коли властивість IsFocused елемента Entry набуде значення true.

Робота тригера полягатиме в установці ряду властивостей елемента Entry. Тут у Entry встановлюється червоний текст:

```
<Setter Property="TextColor" Value="Red" />
```

Таким чином, при отриманні фокусу спрацює тригер, який забарвить текст у червоний колір.

Якщо встановити фокус в іншому місці програми на якийсь інший елемент, то тригер вже не діятиме, а текст в Entry набуде свого стандартного кольору.

Стиль може містити кілька тригерів, і всі вони визначаються елементом Style.Triggers.

Можна визначати тригери властивостей у коді C#.

Приклад 9.2. Визначення тригера властивостей у коді C#.

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        Entry entry = new Entry();

        // define trigger for Entry
        var trigger = new Trigger(typeof(Entry))
        {
            Property = Entry.IsFocusedProperty,
            Value = true
        };
        // set green backgroundColor
        trigger.Setters.Add(new Setter
        {
            Property = Entry.BackgroundColorProperty,
            Value = Color.Green
        });
        // set White TextColor
        trigger.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Color.White
        });
```

```
            // add Trigger
            entry.Triggers.Add(trigger);

            Content = new StackLayout
            {
                Children = { entry }
            };
        }
    }
```

В коді визначається тригер за допомогою об'єкта Trigger. Через властивість Property вказується властивість елемента Entry, яка буде відстежуватись. А за допомогою властивості Value встановлюється значення, при отриманні якого спрацює тригер.

```
var trigger = new Trigger(typeof(Entry))
            {
                Property = Entry.IsFocusedProperty,
                Value = true
            };
```

Тобто коли властивість IsFocused отримає значення true, спрацює тригер.

Далі через колекцію Setters визначаються властивості та їх значення, які встановлюватимуться під час дії тригера – встановлення зеленого кольору фону та білого кольору для введених символів.

Наприкінці тригер додається до колекції Triggers елемента Entry:

```
entry.Triggers.Add(trigger);
```

У результаті при отриманні фокусу текстове поле фарбуватиметься в зелений колір, а введені символи - в білий.

Додавання до тригера додаткових дій

Дія в тригері - це звичайний клас, який успадковується від базового класу TriggerAction і реалізує метод Invoke().

У C# та Xamarin, метод Invoke() використовується для виконання коду в потоці, який має дозвіл керувати користувацьким інтерфейсом, особливо коли треба взаємодіяти з UI з іншого потоку. Метод Invoke() в C# використовується для виклику методу делегата в тому ж потоці, в якому був створений об'єкт делегата. Зазвичай це необхідно, коли потрібно виконати певний код в головному потоці програми з іншого потоку.

Invoke() використовується для безпечної взаємодії між потоками в контексті багатопоточності, дозволяючи новому потоку коректно оновлювати елементи графічного інтерфейсу через основний потік.

Коли об'єкт делегата створюється в одному потоці і передається в інший потік, для його виклику в іншому потоці використовується метод Invoke(). Цей

метод викликає метод, який посилається на делегат, і блокує поточний потік до того часу, поки метод не завершиться.

Визначення наступного класу демонструє, як можна керувати візуальними властивостями елементів у Xamarin.Forms за допомогою тригерів на основі станів елементів, таких як фокусування/розфокусування.

Приклад 9.3. Визначення класу у коді C#.

```
using Xamarin.Forms;

namespace lec7del
{
    public class FocusTriggerAction : TriggerAction<Entry>
    {
        protected override void Invoke(Entry sender)
        {
            if (sender.IsFocused)
                sender.FadeTo(1);  // непрозорий
            else
                sender.FadeTo(0.5);
        }
    }
}
```

Клас FocusTriggerAction є прикладом власної дії тригера (TriggerAction) для елементу введення тексту (Entry). Він розширює функціональність базового класу TriggerAction за типом Entry.

У метод Invoke як параметр передається елемент Entry, до якого застосовується тригер. Метод Invoke, який автоматично викликається, коли тригер активується, використовує FadeTo для зміни прозорості Entry в залежності від того, чи елемент активний (тобто має фокус). Якщо елемент Entry має фокус, він стає повністю непрозорим (FadeTo(1)), що означає 100% видимість. Якщо фокус втрачено, елемент стає наполовину прозорим (FadeTo(0.5)), що робить його наполовину прозорим або "затемненим".

У цьому випадку, викликається дія, коли властивість IsFocused елементу Entry змінюється.

Приклад 9.4. Використання тригерів для автоматичного змінення властивостей елемента Entry при зміні його стану фокусування.

```
using Xamarin.Forms;

namespace lec7del
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
```

```
        {
            Entry entry = new Entry();

            var trigger = new Trigger(typeof(Entry))
            {
                Property = Entry.IsFocusedProperty,
                Value = true
            };
            trigger.Setters.Add(new Setter
            {
                Property = Entry.BackgroundColorProperty,
                Value = Color.Green
            });
            trigger.Setters.Add(new Setter
            {
                Property = Entry.TextColorProperty,
                Value = Color.White
            });

         trigger.EnterActions.Add(new FocusTriggerAction());
         trigger.ExitActions.Add(new FocusTriggerAction());

            entry.Triggers.Add(trigger);

            Content = new StackLayout
            {
                Children = { entry }
            };
        }
    }
}
```

Клас Trigger визначає дві спеціальні властивості: EnterActions (зберігає дії, що застосовуються при спрацьовуванні тригера) та ExitActions (зберігає дії, які виконуються, коли тригер перестає діяти). Причому в ці колекції-властивості передається FocusTriggerAction, що дозволяє включити дію при включенні тригера і відключати при відключенні тригера.

Метод Invoke перевіряє, чи має елемент фокус (IsFocused). Якщо true, застосовується ефект згасання (FadeTo) з поточної прозорості до повної (1.0). Якщо false — згасання до 0.5, що робить елемент більш прозорим.

У результаті цей код створює елемент вводу, який змінює свій вигляд при фокусуванні та зніманні фокусу: він стає зеленим із білим текстом при фокусуванні та згасає до меншої прозорості, коли фокус знімається.

Цей приклад показує, як можна використовувати тригери для динамічної зміни зовнішнього вигляду та поведінки інтерфейсу користувача в Xamarin.Forms на основі подій або зміни станів елементів.

Тригери подій

Тригери подій викликаються у відповідь на події елемента.

В Xamarin події та дії функціонують по-різному. Події — це механізми, що дозволяють класам або об'єктам сповіщати про те, що з ними сталося якась подія. Це може бути щось на кшталт натискання кнопки або зміни тексту. Дії, у свою чергу, це блоки коду, які виконуються у відповідь на ці події; наприклад, зміна візуального стилю кнопки при її натисканні. Основна відмінність полягає в тому, що події сигналізують про зміни стану або дії користувача, тоді як дії — це специфічні реакції, які налаштовуються програмістом для реагування на ці події.

Приклад 9.5. Клас EntryValidation для валідації тексту, введеного в Entry.

```
using System;
using Xamarin.Forms;
namespace AppXamarin
{

    public class EntryValidation : TriggerAction<Entry>
    {
        protected override void Invoke(Entry sender)
        {
            int number;
            if (!Int32.TryParse(sender.Text, out number))
                sender.BackgroundColor = Color.Red;
            else
                sender.BackgroundColor = Color.Default;
        }
    }
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
     }
```

Метод Invoke, який викликається, коли активується відповідний тригер, спробує перетворити текст в Entry на ціле число за допомогою методу Int32.TryParse. Якщо текст не може бути перетворений на ціле число (наприклад, якщо він містить літери або спеціальні символи), то фон Entry змінюється на червоний колір, сигналізуючи про помилку у введенні. Якщо текст успішно перетворюється на число, фоновий колір встановлюється на за замовчуванням (Color.Default), що зазвичай є стандартним фоном.

Приклад 9.6. Застосування класу EntryValidation.

```
<?xml version="1.0" encoding="utf-8" ?>
    <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
                 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                 xmlns:local="clr-
namespace:AppXamarin;assembly=AppXamarin"
                 x:Class="AppXamarin.MainPage">
        <ContentPage.Resources>
            <ResourceDictionary>
                <Style x:Key="entryStyle" TargetType="Entry">
                    <Style.Triggers>
                        <EventTrigger Event="TextChanged">
                            <local:EntryValidation />
                        </EventTrigger>
                    </Style.Triggers>
                </Style>
            </ResourceDictionary>
        </ContentPage.Resources>
        <StackLayout>
            <Entry FontSize="Large" Style="{StaticResource
Key=entryStyle}" />
        </StackLayout>
    </ContentPage>
```

Тригер подій також визначається елементом Style.Triggers, тільки тепер він представлений об'єктом EventTrigger.

Атрибут Event цього об'єкта вказує на подію, при виникненні якої буде викликатись тригер. В даному випадку це подія TextChanged, тобто зміна тексту в полі Entry.

І в самому EventTrigger визначається дія EntryValidation.

У результаті, якщо ввести в поле нецифрові символи, це поле пофарбується в червоний колір.

Контрольні питання

1. Що таке тригер у Xamarin Forms, і як він використовується для зміни властивостей об'єктів?
2. Які існують типи тригерів у Xamarin Forms, і яка між ними різниця?
3. Наведіть приклад використання Property Trigger і поясніть його дію.
4. Як можна використати Event Trigger у додатку Xamarin Forms? Наведіть приклад.
5. Що таке Setter, і як його можна використовувати у стилях та тригерах?
6. Для чого використовуються EnterActions і ExitActions у тригерах? Наведіть приклад їх застосування.

Завдання до лабораторної роботи

Додайте наступні тригери у додаток «Калькулятор» (лабораторна робота №4):

- тригер для попередження про помилки, який змінює колір тексту на червоний, якщо користувач вводить некоректний символ або намагається виконати недопустиму математичну операцію (наприклад, ділення на нуль);

- тригер для створення анімації тексту результату (наприклад, збільшення та зменшення числа), коли відображається новий результат обчислення;

- тригер, який змінює розмір кнопок калькулятора в залежності від того, яка кнопка була натиснута.

ЛІТЕРАТУРА

1. Лозинська О.В., Давидов М.В., Демчук А.Б. Програмне забезпечення мобільних пристроїв. Навчальний посібник. – Л.: Новий світ-2000, 2021. – 218 с.

2. Байдачний С. Windows 10 для C# розробників. Книга 1: [навч. посіб.] – К.: ІТ-книга, 2016. – 230 с.

3. Байдачний С. Windows 10 для C# розробників. Книга 2: [навч. посіб.] – К. : ІТ-книга, 2016. – 312 с.

4. Поляков А. О., Федорченко В. М., Шматко О. В. Аналіз методів технологій розроблення мобільних додатків для платформи Android: навчальний посібник [Електронний ресурс] – Харків: ХНЕУ ім. С. Кузнеця, 2017. – 286 с.

5. Android developers (Розробка для Android) [Електронний ресурс]. – Режим доступу: http://developer.android.com.

6. Ресурси та засоби розробки додатків [Електронний ресурс]. - Режим доступу: https://msdn.microsoft.com/app-development-msdn.

7. Розробка для iPhone [Електронний ресурс] – Режим доступу: https://developer.apple.com/ios/

8. Mustafa T., Sohr K. Understanding the implemented access control policy of android system services with slicing and extended static checking/ International Journal of Information Security. – 2014. – p. 1–20.

9. Nolan G., Cinar O., Truxall D. Android best practices – Springer, 2014. – 222 p.

10. Розробка засобами Xamarin [Електронний ресурс] – Режим доступу: https://dotnet.microsoft.com/en-us/apps/xamarin

11. Розробка засобами Maui [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/uk-ua/dotnet/maui/migration/?view=net-maui-8.0&WT.mc_id=dotnet-35129-website

ДОДАТОК 1

Елементи Xamarin Forms, їх основні властивості та методи

Загальні компоненти інтерфейсу:

1. **BoxView** у Xamarin Forms — це простий контрол для відображення кольорового прямокутника або квадрата.

Властивості:

Color - колір заповнення BoxView.

CornerRadius - радіус закруглення кутів BoxView. Може бути встановлений окремо для кожного кута або однаково для всіх.

WidthRequest, HeightRequest - бажані ширина та висота для BoxView.

HorizontalOptions, VerticalOptions - властивості, які контролюють, як BoxView повинен розміщуватися в батьківському контейнері по горизонталі та вертикалі.

Opacity - прозорість BoxView, де 1 означає повністю непрозорий, а 0 - повністю прозорий.

Методи:

InvalidateMeasure() - інвалідує виміри контролю, спричинюючи їх повторний розрахунок.

OnMeasure(Double, Double) - викликається, коли контрол потребує вимірювання, щоб повідомити розмір відповідно до його вмісту та своїх властивостей.

OnSizeAllocated(Double, Double) - викликається, коли контрол отримує розмір від батьківського контейнера.

Події:

SizeChanged - подія, яка викликається, коли розміри BoxView змінюються.

2. **Button**— це стандартний контрол для створення інтерактивної кнопки, що реагує на натискання.

Властивості Button в Xamarin:

BackgroundColor - колір фону кнопки.

BorderRadius - радіус скруглення кутів кнопки.

BorderColor - колір межі кнопки.

BorderWidth - ширина межі кнопки.

Command - команда, яка виконується при натисканні на кнопку.

CommandParameter - параметр, який передається команді при її виконанні.

CornerRadius - застосовує круглість до кутів кнопки.

FontAttributes - атрибути шрифту тексту кнопки, такі як жирний, курсив тощо.

FontFamily - назва шрифту тексту кнопки.

FontSize - розмір шрифту тексту кнопки.

ImageSource - зображення, яке відображається на кнопці.

IsEnabled - вказує, чи активна кнопка для натискання.

Text - текст, який відображається на кнопці.

TextColor - колір тексту кнопки.

Методи Button в Xamarin:

Focus() - переміщує фокус на цю кнопку.

SendClicked() - викликає обробник події Clicked для кнопки.

SetBinding(BindableProperty, BindingBase) - встановлює прив'язку даних для вказаної властивості кнопки.

Unfocus() - прибирає фокус з кнопки.

Події Button:

Clicked - подія спрацьовує, коли користувач натискає на кнопку.

Pressed - подія викликається, коли кнопка стає натиснутою. Вона спрацьовує в момент натискання на кнопку, без очікування відпускання користувачем.

Released - подія викликається, коли кнопка була натиснута та потім відпущена. Вона спрацьовує після того, як користувач відпустив кнопку після натискання.

3. **Label** у Xamarin Forms — це елемент інтерфейсу, що використовується для відображення тексту у додатку.

Властивості:

Text - текст, що відображається у Label.

TextColor - колір тексту.

BackgroundColor - колір фону мітки.

FontFamily, FontSize, FontAttributes - властивості, що керують шрифтом тексту.

HorizontalTextAlignment, VerticalTextAlignment - вирівнювання тексту по горизонталі та вертикалі в межах контролу.

LineBreakMode - визначає, як текст повинен бути обрізаний або обгорнутий на кінцях рядків.

FormattedText - дозволяє використовувати багатоформатний текст.

Padding - внутрішній відступ навколо тексту всередині Label.

HorizontalOptions - визначає, як мітка вирівнюється по горизонталі всередині свого контейнера.

VerticalOptions - визначає, як мітка вирівнюється по вертикалі всередині свого контейнера.

Методи Label у Xamarin:

InvalidateMeasure() - інвалідує виміри контролу, спричинюючи їх повторний розрахунок.

OnMeasure(Double, Double) - викликається, коли контрол потребує вимірювання, щоб повідомити розмір відповідно до його вмісту та своїх властивостей.

OnSizeAllocated(Double, Double) - викликається, коли контрол отримує розмір від батьківського контейнера.

Події:

SizeChanged - виникає, коли змінюється розмір Label.

Focused і Unfocused - виникають, коли Label отримує або втрачає фокус, хоча Label зазвичай не взаємодіє з користувачем як інтерактивний елемент.

4. **Entry** — це елемент управління, що дозволяє користувачам вводити короткий текст через однорядкове текстове поле.

Властивості:

Text - актуальний текст у полі вводу.

BackgroundColor - колір фону вводу.

CharacterSpacing - міжсимвольний інтервал вводу.

FontAttributes, FontFamily, FontSize - налаштування шрифту.

HorizontalTextAlignment - горизонтальне вирівнювання тексту вводу.

IsReadOnly - показує, чи є введення лише для читання.

IsPassword - чи слід приховувати введений текст для конфіденційності, наприклад, при введенні пароля.

Keyboard - тип клавіатури, який має бути використаний, наприклад, для чисел, електронної пошти тощо.

MaxLength - максимальна кількість символів, дозволених для введення.

Placeholder - текст-підказка, що відображається в Entry, коли воно порожнє.

TextColor, PlaceholderColor - кольори для тексту та тексту-підказки.

VerticalTextAlignment - вертикальне вирівнювання тексту вводу.

Методи:

Focus() - переміщає фокус на це поле вводу.

OnTextChanged(String, String) - викликається, коли текст у полі вводу змінюється.

Unfocus() - прибирає фокус з цього поля вводу.

Події:

Completed - викликається, коли користувач завершує введення тексту, зазвичай після натискання кнопки "Enter" на клавіатурі.

Focused - подія викликається, коли поле вводу отримує фокус.

TextChanged - спрацьовує при зміні тексту в Entry.

TextChangedCommand - подія дозволяє зв'язати команду зі зміною тексту в полі вводу.

Unfocused - подія викликається, коли поле вводу втрачає фокус.

5. **Editor** у Xamarin Forms — це контрол (редактор) для введення тексту, який забезпечує багаторядкове текстове поле.

Властивості Editor в Xamarin:

Text - текст, що вводиться, або введений в редактор.

AutoSize - визначає, чи повинен Editor автоматично змінювати свій розмір, щоб вмістити текст.

BackgroundColor - колір фону редактора.

CharacterSpacing - міжсимвольний інтервал редактора.

FontAttributes - атрибути шрифту редактора, такі як жирний, курсив тощо.

FontFamily - назва шрифту для редактора.

FontSize - розмір шрифту редактора.

IsReadOnly - якщо true, користувач не може змінювати текст.

Keyboard - тип клавіатури для використання.

MaxLength - максимальна дозволена кількість символів.

Placeholder - текст-підказка, який відображається, коли редактор порожній.

PlaceholderColor - колір тексту підказки.

TextColor - колір тексту редактора.

Методи Editor в Xamarin:

Focus() - переміщає фокус на цей редактор.

OnTextChanged(String, String) - викликається, коли текст у редакторі змінюється.

ScrollTo(int, int, bool) - прокручує редактор до вказаного місця.

Unfocus() - прибирає фокус з цього редактора.

Події Editor:

Completed - подія викликається, коли користувач завершує введення тексту і натискає на клавішу "Готово" або використовує подію клавіатури "Enter".

Focused - подія викликається, коли редактор отримує фокус.

TextChanged - подія спрацьовує, коли текст у редакторі змінюється.

TextChangedCommand - подія дозволяє зв'язати команду зі зміною тексту у редакторі.

Unfocused - подія викликається, коли редактор втрачає фокус.

6. **Image** — це віджет для відображення зображень у мобільному додатку. Він підтримує різні формати зображень, такі як JPEG, PNG, GIF, BMP та інші, і дозволяє легко інтегрувати зображення з локальних файлів, URL-адрес або вбудованих ресурсів.

Властивості:

Source - властивість, яка визначає джерело зображення. Це може бути URI до зображення в Інтернеті, локальний шлях до файлу в системі або ресурс, вбудований в додаток. Для зазначення джерела можна використовувати клас ImageSource.

Aspect - визначає, як зображення буде масштабуватися та обрізатися, щоб відповідати визначеним розмірам Image. Опції включають AspectFit (зберігає пропорції, вміщуючись повністю в доступний простір), AspectFill (зберігає пропорції, заповнюючи весь доступний простір та обрізаючи зайве) та Fill (заповнює весь доступний простір, ігноруючи пропорції).

IsLoading - властивість, яка повертає true, коли зображення завантажується.

IsOpaque - встановлення цієї властивості допомагає оптимізувати рендеринг у деяких сценаріях, зазначивши, що зображення не має прозорих областей.

Opacity - властивість для встановлення прозорості зображення, де 1 означає повністю непрозоре, а 0 - повністю прозоре.

BackgroundColor - колір фону зображення.

BorderColor - колір межі зображення.

BorderWidth - ширина межі зображення.

HeightRequest - вимога щодо висоти зображення.

HorizontalOptions - визначає, як зображення вирівнюється по горизонталі всередині свого контейнера.

VerticalOptions - визначає, як зображення вирівнюється по вертикалі всередині свого контейнера.

WidthRequest - вимога щодо ширини зображення.

Методи Image у Xamarin:

InvalidateMeasure() - інвалідує виміри контролу, спричинюючи їх повторний розрахунок.

OnMeasure(Double, Double) - викликається, коли контрол потребує вимірювання, щоб повідомити розмір відповідно до його вмісту та своїх властивостей.

OnSizeAllocated(Double, Double) - викликається, коли контрол отримує розмір від батьківського контейнера.

7. Клас **ImageView** надає доступ до зображень у мобільному додатку.

Image використовується у Xamarin.Forms для кросплатформної розробки, а ImageView - це компонент Android, який використовується в Xamarin.Android для специфічних випадків на цій платформі.

Властивості:

src - вказує ресурс або шлях до зображення, яке має відображатися в ImageView.

scaleType - визначає тип масштабування зображення в ImageView (наприклад, "fitCenter", "centerCrop", тощо).

adjustViewBounds - дозволяє ImageView автоматично міняти свої межі, щоб відповідати пропорціям зображення, яке він відображає, якщо scaleType встановлено на "fitXY".

visibility - визначає видимість ImageView ("visible", "invisible", "gone").

alpha - вказує прозорість зображення у діапазоні від 0 (прозорий) до 1 (повністю видимий).

Методи:

setImageResource(int resId) - встановлює ресурс зображення для ImageView на основі ідентифікатора ресурсу.

setImageDrawable(Drawable drawable) - встановлює Drawable об'єкт як зображення для ImageView.

setOnClickListener(View.OnClickListener listener) - встановлює обробник кліків на ImageView.

getDrawable() - повертає Drawable об'єкт, що представляє поточне зображення ImageView.

setScaleType(ScaleType scaleType) - встановлює тип масштабування для зображення в ImageView.

setVisibility(int visibility) - встановлює видимість ImageView.

setAlpha(float alpha) - встановлює прозорість зображення в ImageView.

Події ImageView:

onClick - спрацьовує, коли користувач клікає на зображення.

onLongClick - спрацьовує, коли користувач довго утримує натискання на зображенні.

onTouch - спрацьовує, коли користувач торкається зображення.

Специфічні контейнери і лейаути:

8. **Frame** у Xamarin Forms є контейнером, що використовується для організації вмісту з можливістю додавання рамки навколо дочірнього елемента.

Властивості:

BorderColor - визначає колір рамки Frame.

BorderWidth - ширина межі рамки.

CornerRadius - задає радіус закруглення кутів рамки.

HasShadow - керує тим, чи має Frame тінь.

Content - дочірній елемент, який відображається всередині Frame.

Padding - внутрішні відступи між рамкою і дочірнім елементом.

BackgroundColor - колір фону Frame.

IsClippedToBounds - вказує, чи вирізається контент рамки за її межі.

Методи:

Add (VisualElement) - додає дочірній елемент в Frame.

Remove (VisualElement) - видаляє дочірній елемент з Frame.

Clear() - очищує дочірні елементи всередині Frame.

InvalidateMeasure() - інвалідує виміри контролю, спричинюючи їх повторний розрахунок.

OnMeasure(Double, Double) - викликається, коли контрол потребує вимірювання, щоб повідомити розмір відповідно до його вмісту та своїх властивостей.

OnSizeAllocated(Double, Double) - викликається, коли контрол отримує розмір від батьківського контейнера.

Події:

SizeChanged - виникає, коли змінюється розмір Frame.

Focused та Unfocused - виникають, коли Frame отримує або втрачає фокус.

9. **LayoutOptions** у Xamarin Forms – це структура для налаштування розташування елементів у макеті. Вона визначає, як елемент має взаємодіяти з батьківським контейнером, описує, як елемент повинен розміщуватися та розтягуватися всередині його контейнера.

Властивості:

Expands - логічне значення, що вказує, чи має елемент розтягуватися, щоб зайняти додатковий доступний простір у своєму контейнері.

Alignment - вказує, як елемент вирівнюється у своєму розподільному просторі (може бути

Start - елемент розташовується з початку контейнера. У горизонтальному напрямку це відповідає лівому краю, а у вертикальному - верхньому краю.

Center - елемент розташовується по центру контейнера.

End - елемент розташовується в кінці контейнера. У горизонтальному напрямку це відповідає правому краю, а у вертикальному - нижньому краю.

Fill - елемент заповнює весь доступний простір у батьківському контейнері.

StartAndExpand - елемент розташовується з початку контейнера і розтягується на всю доступну ширину або висоту, залежно від орієнтації.

CenterAndExpand - елемент розташовується по центру контейнера і розтягується на всю доступну ширину або висоту, залежно від орієнтації.

EndAndExpand - елемент розташовується в кінці контейнера і розтягується на всю доступну ширину або висоту, залежно від орієнтації.)

Використання:

LayoutOptions застосовується через властивості такі як HorizontalOptions та VerticalOptions у багатьох візуальних контролах.

10. **StackLayout** — це контейнер, що використовується для розташування дочірніх елементів у вертикальному або горизонтальному порядку.

Властивості:

Children - колекція елементів, які розміщуються в StackLayout.

HorizontalOptions та VerticalOptions - визначають розташування StackLayout в батьківському контейнері.

Orientation - визначає орієнтацію StackLayout. Може бути Horizontal (горизонтально) або Vertical (вертикально).

Spacing - визначає відстань між дочірніми елементами.

Методи StackLayout:

Add - додає дочірній елемент до StackLayout.

Clear - видаляє всі дочірні елементи з StackLayout.

Remove - видаляє дочірній елемент з StackLayout.

RemoveAt - видаляє дочірній елемент з вказаною позицією в StackLayout.

11. **Grid** – це контейнерний макет для розміщення елементів у вигляді сітки, який дозволяє організувати їх у рядки та стовпці.

Властивості:

RowDefinitions - визначає розміри та властивості рядків у сітці.

ColumnDefinitions - визначає розміри та властивості стовпців у сітці.

Children - колекція, що містить дочірні елементи, які додаються до Grid.

Grid.Row і Grid.Column - властивості, які встановлюють, у якому рядку чи стовпці має розташовуватися дочірній елемент.

Методи Grid:

Add (VisualElement, int, int) - додає елемент до Grid у вказану позицію (рядок, стовпець).

Remove (VisualElement) - видаляє елемент з Grid.

Clear() - очищує всі дочірні елементи з Grid.

SetRow (BindableObject, int) та SetColumn (BindableObject, int) - встановлюють рядок і стовпець для дочірнього елемента в Grid.

SetRowSpan (BindableObject, int) та SetColumnSpan (BindableObject, int) - встановлюють, скільки рядків або стовпців повинен займати елемент.

12. **RelativeLayout** — це контейнер, який дозволяє розміщувати дочірні елементи відносно самого контейнера або інших дочірніх елементів.

Властивості:

XConstraint та YConstraint - визначають горизонтальне і вертикальне розташування елемента відносно інших елементів або батьківського контейнера.

WidthConstraint та HeightConstraint - визначають ширину і висоту елемента.

RelativeLayout.XConstraintProperty, RelativeLayout.YConstraintProperty - дозволяють встановлювати значення для XConstraint та YConstraint у коді.

RelativeLayout.WidthConstraintProperty, RelativeLayout.HeightConstraintProperty - дозволяють встановлювати значення для WidthConstraint та HeightConstraint у коді.

RelativeLayout.Behaviors - колекція додаткових функцій, які можуть бути застосовані до елементів у RelativeLayout.

Методи RelativeLayout:

AddConstraint - додає обмеження до елемента у RelativeLayout.

RemoveConstraint - видаляє обмеження з елемента у RelativeLayout.

GetXConstraint та GetYConstraint - повертає обмеження по горизонталі і вертикалі для вказаного елемента.

GetWidthConstraint та GetHeightConstraint - повертає обмеження ширини і висоти для вказаного елемента.

Таблиці та колекції:

13. **TableView**— це контейнер, призначений для структурованого відображення даних у формі таблиці, для відображення статичної, структурованої інформації. TableView містить рядки, поділені на секції. Елементи управління у TableView можуть бути організовані за допомогою класів TableSection, TextCell, SwitchCell та інших.

Властивості TableView:

Intent - загальний вигляд TableView. Значення може бути Form, Menu, Settings або Data, кожен з яких надає таблиці відповідний стиль та поведінку для різних сценаріїв використання.

Root - властивість, яка містить колекцію TableSection. Root є контейнером для всіх секцій у TableView. Кожна TableSection може містити одну або більше клітинок (cells), які представляють собою рядки таблиці.

HasUnevenRows - якщо встановлено true, це дозволяє рядкам мати різну висоту в залежності від вмісту.

RowHeight - встановлює фіксовану висоту для всіх рядків у TableView. Якщо HasUnevenRows встановлено як true, ця властивість ігнорується.

IsEnabled - керує доступністю TableView, дозволяючи вимкнути взаємодію з користувачем, якщо встановлено false.

BackgroundColor - колір фону TableView.

BindingContext - контекст прив'язки даних для TableView та всіх його дочірніх елементів.

Методи TableView у Xamarin:

RemoveBinding(BindableObject, BindableProperty) - видаляє прив'язку даних з вказаного властивості елемента.

SetBinding(BindableObject, BindableProperty, BindingBase) - встановлює прив'язку даних для вказаного властивості елемента.

OnBindingContextChanged()- викликається, коли змінюється контекст прив'язки даних TableView.

Утиліти і службові класи:

14. **ResourceDictionary** у Xamarin Forms — це контейнер для зберігання повторно використовуваних ресурсів, таких як стилі, кольори, шрифти та інші об'єкти, які можуть бути використані в різних місцях додатку. Це забезпечує централізоване місце для управління ресурсами, що спрощує зміни в дизайні та забезпечує більшу консистентність візуального представлення.

Властивості:

Keys - колекція ключів ресурсів у словнику.

MergedDictionaries - колекція, що дозволяє злити декілька ResourceDictionaries в один.

Source - вказує на зовнішній ресурс, з якого можна завантажити додаткові визначення.

Методи:

Add(key, value) - додає новий ресурс з вказаним ключем і значенням до словника.

Remove(key) - видаляє ресурс за вказаним ключем.

Clear() - очищує всі ресурси в словнику.

ContainsKey(key) - перевіряє, чи містить словник ресурс із заданим ключем.

SetValue(Object, Object) - встановлює значення для вказаного ключа в ResourceDictionary.

TryGetValue(key, out value) - пробує отримати значення ресурсу за ключем, повертає bool, який вказує на успішність операції.

ResourceDictionary не є візуальним елементом, він не має власних подій. Вміст ResourceDictionary може бути використаний в інших частинах додатку через ресурси.

OnResourceChanged - ця подія спрацьовує, коли ресурс в ResourceDictionary змінюється.

15. **Клас Setter в Xamarin**

використовується для встановлення значень властивостей об'єктів.

Властивості:

Property - назва властивості, яку потрібно встановити.

Value - значення, яке потрібно встановити для вказаної властивості.

Методи:

Equals(Object) - порівнює об'єкт Setter з іншим об'єктом.

GetHashCode() - повертає хеш-код об'єкта Setter.

ToString() - повертає рядок, який представляє об'єкт Setter.

Клас Setter використовується для встановлення значень властивостей елементів у стилях і тригерах. Наприклад, за допомогою класу Setter можна встановити значення текстового кольору або розміру шрифту для елемента у стилі.

16. Клас **Style** у Xamarin.Forms — це механізм, який дозволяє визначити візуальні характеристики компонентів інтерфейсу (таких як шрифти, кольори тощо) в одному місці і застосовувати їх послідовно до різних елементів аплікації. Стиль задається через клас Style.

Властивості:

TargetType вказує на клас, до якого застосовується стиль.

Setters використовується для визначення колекції Setter об'єктів, кожен з яких задає значення певної властивості для цільового елемента. Це дозволяє централізовано керувати властивостями елементів, які треба стилізувати.

BasedOn дозволяє спадкувати стиль від іншого стилю, дозволяє новому стилю наслідувати і розширювати існуючий стиль.

Triggers - властивість Triggers об'єкта Style в Xamarin використовується для визначення тригерів, які активуються в залежності від певних умов і виконують певні дії.

Властивість

Count - повертає кількість тригерів у колекції.

Методи, які пов'язані з колекцією тригерів в Xamarin, а саме з властивістю Triggers об'єкта Style:

Clear() - видаляє всі тригери з колекції.

Remove(TriggerBase) - видаляє вказаний тригер з колекції.

Add(TriggerBase) - додає тригер в колекцію.

Основи сторінок та навігації:

17. **Page** у Xamarin Forms — це базовий компонент для всіх видів сторінок у додатку. Це контейнер, що містить інтерфейс користувача додатку. Сторінка може відображати лейаути, контроли та інші візуальні елементи.

Властивості Page в Xamarin:

BackgroundColor - колір фону сторінки.

BindingContext - контекст прив'язки даних для сторінки та всіх її дочірніх елементів.

Content - вміст сторінки, який включає всі елементи користувальницького інтерфейсу.

IconImageSource - зображення, яке використовується як значок на панелі навігації або вкладці сторінки.

IsBusy - індикатор активності, який можна використовувати для показу статусу завантаження чи обробки.

Padding - відступи вмісту сторінки від країв її контейнера.

Title - заголовок сторінки, який відображається у рядку заголовка області перегляду.

ToolbarItems- колекція елементів, які з'являються в панелі інструментів сторінки.

Методи Page в Xamarin:

LayoutChanged() - метод викликається, коли розміщення сторінки змінюється.

OnSizeAllocated() - метод викликається, коли розмір сторінки змінюється.

Події об'єкту Page:

Appearing - виникає, коли сторінка відображається перед користувачем.

Disappearing - виникає, коли сторінка зникає з видимості користувача.

LayoutChanged - спрацьовує, коли розміщення сторінки змінюється.

SizeChanged - виникає, коли розмір сторінки змінюється (це стосується змін у розмірах, які можуть виникати при зміні орієнтації пристрою або інших факторах, що впливають на розмір відображення).

PropertyChanged - виникає, коли змінюється властивість об'єкта сторінки, використовується для відстеження змін властивостей, які викликаються зовнішніми факторами або логікою додатку.

ДОДАТОК 2

Список кодів помилок та попереджень при розробці за допомогою Xamarin.Android

ADBxxxx: Інструменти ADB:

- **ADB0000: Загальна помилка або попередження в ADB.**
- **ADB0010**: Загальна помилка або попередження при встановленні APK.
- **ADB0020**: Пакет не підтримує архітектуру процесора даного пристрою.
- **ADB0030**: Збій встановлення APK через конфлікт з існуючим пакетом.
- **ADB0040**: Пристрій не підтримує мінімальний рівень SDK, зазначений у маніфесті додатка.
- **ADB0050**: Пакет {packageName} вже існує на пристрої.
- **ADB0060**: Недостатньо місця на пристрої для зберігання пакета {packageName}. Звільніть місце та спробуйте знову.

ANDXXxxxx: Універсальний інструментарій Android:

- **ANDAS0000:** Загальна помилка або попередження в apksigner.
- **ANDJS0000**: Загальна помилка або попередження в jarsigner.
- **ANDKT0000**: Загальна помилка або попередження в keytool.
- **ANDZA0000**: Загальна помилка або попередження в zipalign.

APTxxxx: Інструменти AAPT:

- **APT0000:** Загальна помилка або попередження в AAPT.
- **APT0001:** Невідома опція {option} у командному рядку AAPT.

XA0xxx: Проблеми з навколишнім середовищем або відсутність інструменту:

- **XA0000**: Не вдалося визначити $(AndroidApiLevel) або $(TargetFrameworkVersion).
- **XA0001**: Недопустиме або непідтримуване значення $(TargetFrameworkVersion).
- **XA0002**: Не вдалося знайти mono.android.jar.
- **XA0030**: Збірка застарілої версії JDK {versionNumber} не підтримується.
- **XA0031**: Для націлення на FrameworkVersion {targetFrameworkVersion} потрібно Java SDK версії {requiredJavaForFrameworkVersion} або новішої.
- **XA0032**: При використанні інструментів збірки {buildToolsVersion} потрібен пакет SDK для Java версії {requiredJavaForBuildTools} або новішої.

- **XA0033**: Не вдалося отримати версію Java SDK, оскільки вона не містить дійсного номера версії.
- **XA0034**: Не вдалося отримати версію пакета SDK для Java.
- **XA0100**: Недопустима бібліотека EmbeddedNativeLibrary у проекті додатка Android. Замість цього використовуйте AndroidNativeLibrary.
- **XA0101**: Попередження XA0101: дія збірки @(Content) не підтримується.
- **XA0102**: Загальне попередження lint.
- **XA0103**: Загальна помилка lint.
- **XA0104**: Недопустимий режим точки слідування.
- **XA0105**: $(TargetFrameworkVersion) для бібліотеки DLL більше, ніж $(TargetFrameworkVersion) для проекту.
- **XA0107**: Є еталонною збіркою {Assmebly}.
- **XA0108**: Не вдалося отримати версію з lint.
- **XA0109**: Непідтримуване або недопустиме значення v4.5.$(TargetFrameworkVersion).
- **XA0110**: Відключення $(AndroidExplicitCrunch), оскільки він не підтримується . Якщо ви хочете використовувати $(AndroidExplicitCrunch), встановіть $(AndroidUseAapt2) у false.
- **XA0111**: Не вдалося отримати версію aapt2. Будь ласка, переконайтеся, що він встановлений правильно.
- **XA0112**: Не встановлений aapt2. Відключення підтримки. Будь ласка, переконайтеся, що aapt2 встановлений правильно.
- **XA0113**: Google Play вимагає, щоб нові додатки та оновлення використовували TargetFrameworkVersion версії 8.0 (API рівня 26) або вище.
- **XA0114**: Google Play вимагає, щоб у оновленнях додатків використовувалася версія 8.0 (API рівня 26) або вище. $(TargetFrameworkVersion).
- **XA0115**: Недопустиме значення "armeabi" у $(AndroidSupportedAbis). Цей ABI більше не підтримується. Будь ласка, оновіть властивості проекту, щоб видалити старе значення. Якщо на сторінці властивостей не відображається прапорець 'armeabi', зніміть та знову встановіть один з інших ABI та збережіть зміни.
- **XA0116**: Не вдається знайти ім'я EmbeddedResource{ResourceName}.
- **XA0117**: TargetFrameworkVersion {TargetFrameworkVersion} застаріло. Будь ласка, оновіть його до версії 4.4 або вище.
- **XA0118**: Не вдалося проаналізувати '{TargetMoniker}'.

XA1xxx: Пов'язані з проектом:

- **XA1000:** Проблема з синтаксичним аналізом {file}.
- **XA1001:** AndroidResgen: попередження при оновленні XML ресурсу '{filename}': {message}.

- **XA1002:** Знайдений відповідний ключ '{Key}' для '{Item}'. Але корпус був неправильний. Будь ласка, виправте корпус.
- **XA1003**: '{zip}' не існує. Будь ласка, перебудуйте проект.
- **XA1004**: Виникла помилка при відкритті {filename}. Можливо, файл пошкоджений. Спробуйте видалити його і знову збудувати.
- **XA1005**: Спроба виправлення наївного імені типу для елемента з ідентифікатором '{id}' та типом '{managedType}'.
- **XA1006**: Додаток працює на більш пізній версії Android ({compileSdk}), ніж зазначено у targetSdkVersion ({targetSdk}). Задайте для targetSdkVersion останню доступну версію Android, щоб вона відповідала TargetFrameworkVersion ({compileSdk}).
- **XA1007**: Значення minSdkVersion ({minSdk}) більше, ніж targetSdkVersion. Змініть значення таким чином, щоб minSdkVersion був менше або рівний targetSdkVersion ({targetSdk}).
- **XA1008**: TargetFrameworkVersion ({compileSdk}) не повинна бути нижчою, ніж targetSdkVersion ({targetSdk}).
- **XA1009**: {assembly} застаріла. Будь ласка, оновіться до {assembly} {version}.

XA4xxx: Генерація коду:

- **XA4214**: Управлінський тип "Library1.Class1" існує у декількох збірках: Library1, Library2. Виконайте рефакторинг назв управлінських типів у цих збірках, щоб вони не були ідентичними.
- **XA4215**: Тип Java com.contoso.library1.Class1 створюється декількома управлінськими типами. Будь ласка, змініть атрибут [Register] так, щоб не виходив один і той же тип Java.
- **XA4216**: AndroidManifest.xml //uses-sdk/@android:minSdkVersion '{min_sdk?Value}' менше, ніж API-{XABuildConfig.NDKMinimumApiAvailable}, ця конфігурація не підтримується.
- **XA4218**: Не вдається знайти //manifest/application/uses-library по шляху: {path}.
- **XA4301**: Apk вже містить елемент .xxx.
- **XA4302**: Злиття необроблених винятків "AndroidManifest.xml]": {ex}.
- **XA4303**: Помилка при витягу ресурсів з "{assemblyPath}": {ex}.
- **XA4304**: Файл конфігурації Proguard "{file}" не знайдено.
- **XA4305**: MultiDex увімкнено, але '{nameof (MultiDexMainDexListFile)}' не вказано.

XA5xxx: GCC та набір інструментів:

- **XA5205:** Не вдається знайти в Android SDK.{ToolName}.
- **XA5300:** Не вдалося знайти каталог Android/Java SDK.

Навчальне видання

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОБІЛЬНИХ ПРИСТРОЇВ. Частина 2

електронні методичні вказівки до лабораторних робіт студентам
факультету математики, фізики та інформаційних технологій
першого (бакалаврського) рівня освіти,
галузі 12 «Інформаційні технології»

Електронне практичне видання

Укладачі

Юрій Олександрович Гунченко

Алла Вікторівна Камєнєва

Оксана Миколаївна Зуй

Єгор Олексійович Зудіхін

В авторській редакції

Затвердж. авт. 01.07.2024. Шрифт Times New Roman.
Системні вимоги:
операційна система сумісна з програмним забезпеченням
для читання файлів формату PDF.
Обсяг даних 1,23 Мб. Замовл. №

Видавець і виготовлювач
Одеський національний університет імені І. І. Мечникова
Свідоцтво суб'єкта видавничої справи ДК № 4215 від 22.11.2011 р.
65082, м. Одеса, вул. Єлісаветинська, 12, Україна
Тел.: (048) 723 28 39, e-mail: druk@onu.edu.ua