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

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОБІЛЬНИХ ПРИСТРОЇВ

Частина 1

Методичні вказівки до лабораторних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальностей 122 «Комп'ютерні науки»,
123 «Комп'ютерна інженерія», 151 «Автоматизація та
комп'ютерно-інтегровані технології»

ОДЕСА
ОЛДІ+
2024

УДК 004.4’2’23:621.395.721.5(076.5)
П784

Укладачі:
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, асистент викладача Віденського технічного університету прикладних наук

Рецензенти:
А. Рачинська, канд. фіз.-мат. наук, доцент, зав. кафедрою механіки, автоматизації та інформаційних технологій Одеського національного університету імені І. І. Мечникова;

Ю. Гунченко, док. тен. наук, проф, зав. кафедрою комп'ютерних систем та технологій Одеського національного університету імені І. І. Мечникова

Рекомендовано вченою радою
факультету математики, фізики та інформаційних технологій
Одеського національного університету імені І. І. Мечникова
Протокол № 9 від 27 червня 2024 р.

П784 **Програмне** забезпечення мобільних пристроїв. Частина 1 : методичні вказівки до лабораторних робіт для здобувачів першого (бакалаврського) рівня вищої освіти спеціальностей 122 «Комп’ютерні науки», 123 «Комп’ютерна інженерія», 151 «Автоматизація та комп’ютерно-інтегровані технології» / уклад.: А. В. Камєнєва, О. М. Зуй, Є. О. Зудіхін. – Одеса : ОЛДІ+, 2024. – 68 с.

Методичні вказівки призначені для виконання лабораторних робіт з дисципліни «Програмне забезпечення мобільних пристроїв» для здобувачів першого (бакалаврського) рівня вищої освіти спеціальностей 122 «Комп’ютерні науки», 123 «Комп’ютерна інженерія», 151 «Автоматизація та комп’ютерно-інтегровані технології» факультету математики, фізики та інформаційних технологій Одеського національного університету імені І. І. Мечникова. Кожна лабораторна робота містить теоретичні відомості, приклади, контрольні запитання та завдання для самостійної роботи студентів.

УДК 004.4’2’23:621.395.721.5(076.5)

ЗМІСТ

ВСТУП

Дані методичні вказівки до лабораторних занять підготовлені відповідно до програми курсу «Програмне забезпечення мобільних пристроїв». Навчальна дисципліна відноситься до дисциплін вільного вибору студентів 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, навичок тестування та налагодження мобільних додатків, а також підвищення надійності та безпеки додатків.

Пропоновані Вашій увазі методичні вказівки є першою частиною методичних вказівок до лабораторних занять з 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 №1
НАЛАШТУВАННЯ СЕРЕДОВИЩА РОЗРОБКИ З XAMARIN У VISUAL STUDIO ENTERPRISE 2022

Мета роботи: ознайомитись із процесом налаштування середовища розробки для створення мобільних додатків за допомогою Xamarin у Visual Studio Enterprise 2022, включаючи встановлення необхідних компонентів, конфігурацію платформи та створення тестового проекту для перевірки правильності налаштувань.

Теоретичні відомості

Xamarin є потужною та гнучкою технологією для розробки крос-платформних мобільних додатків, яка дозволяє розробникам використовувати C# та .NET для створення додатків, що працюють на різних мобільних платформах, включаючи Android, iOS та Windows. Використання Xamarin дозволяє значно скоротити час та витрати на розробку, оскільки розробники можуть використовувати спільний код для різних платформ, тим самим забезпечуючи консистентність та ефективність розробки.

Основою Xamarin є Mono, відкритий фреймворк, що розширює можливості .NET Framework на інші платформи за межами Windows. Mono дозволяє запускати додатки, написані на C#, на різних операційних системах, включаючи Linux, macOS, і, звичайно ж, мобільні ОС, такі як Android і iOS. Xamarin використовує Mono для створення мобільних додатків, що дозволяє розробникам використовувати .NET та C# для створення нативних мобільних додатків, забезпечуючи високу продуктивність та доступ до всіх нативних API платформ.

Xamarin відіграє ключову роль у світі розробки мобільних додатків, забезпечуючи розробникам інструменти для створення високоякісних, ефективних та крос-платформних мобільних додатків за допомогою знайомих та потужних технологій, таких як C# та .NET Framework. Це ставить Xamarin у вигідне становище у порівнянні з іншими підходами до крос-платформної розробки, оскільки він забезпечує баланс між продуктивністю, доступом до нативних можливостей платформи та можливістю використання спільного коду.

Архітектура Xamarin організована таким чином, що дозволяє розробникам використовувати спільну логіку додатку між різними платформами, одночасно забезпечуючи можливість додавання платформо-специфічного коду для досягнення нативної продуктивності та вигляду додатків.

Xamarin.Android і Xamarin.iOS є двома ключовими компонентами Xamarin, що дозволяють розробляти нативні мобільні додатки для відповідних платформ.

Xamarin.Android дозволяє розробникам писати код на C#, який компілюється у виконуваний код для Android. Xamarin використовує Mono для запуску цього коду на Android-пристроях, дозволяючи доступ до всіх Android API та функцій платформи через C#.

Xamarin.iOS діє аналогічно для iOS, дозволяючи розробникам використовувати C# та .NET для створення додатків, які нативно працюють на iOS-пристроях. Xamarin.iOS використовує Ahead-Of-Time (AOT) компіляцію для перетворення C# коду у нативний код, який може виконуватися на iOS.

Спільна логіка додатку в Xamarin реалізована через Shared Projects або Portable Class Libraries (PCLs), дозволяючи розробникам писати код логіки додатку один раз і використовувати його на всіх платформах. Це може включати бізнес-логіку, взаємодію з мережею, маніпуляції з даними та інше.

Платформо-специфічний код використовується для досягнення нативного вигляду та відчуття додатку на кожній платформі, включаючи використання нативних елементів інтерфейсу та доступ до функцій, що є унікальними для кожної платформи. У Xamarin платформо-специфічний код зазвичай реалізується через Xamarin.Forms для спільних елементів UI або через платформо-специфічні проекти, що дозволяє використовувати нативні SDK та API для досягнення найкращого користувацького досвіду.

Архітектура Xamarin забезпечує гнучкість та потужність для розробки мобільних додатків, комбінуючи переваги спільної логіки додатку з можливістю використання платформо-специфічного коду для кожної цільової платформи.

Visual Studio Enterprise 2022, як інтегроване середовище розробки (IDE), пропонує широкий спектр можливостей для розробників мобільних додатків, які використовують Xamarin, включаючи інструменти для розробки, налагодження та тестування. Ось огляд ключових можливостей:

Підтримка крос-платформної розробки: Visual Studio Enterprise 2022 дозволяє розробляти, налагоджувати та тестувати мобільні додатки для iOS, Android та Windows з одного проекту, що значно спрощує процес розробки.

Xamarin.Forms: Ця бібліотека дозволяє розробникам створювати інтерфейси користувача, які працюють на кількох платформах за допомогою XAML та C#. Visual Studio надає візуальні дизайнери та інструменти для редагування XAML, що полегшує розробку інтерфейсів.

Потужні інструменти налагодження: IDE містить розширені можливості для налагодження коду, включаючи точки зупину, перегляд стеку викликів, оцінку виразів та змінних у реальному часі, а також специфічні для платформи інструменти налагодження.

Інтеграція з Azure: Розробники можуть легко інтегрувати свої мобільні додатки з хмарними сервісами Azure, використовуючи Azure Mobile Apps для додавання хмарної логіки, автентифікації та зберігання даних.

Інструменти для тестування: Visual Studio включає інструменти для юніт-тестування та інтеграційного тестування, а також підтримку Xamarin Test Cloud для автоматизації тестування UI на різних мобільних пристроях.

Інтеграція з Git: Visual Studio має вбудовану підтримку Git для контролю версій, що дозволяє розробникам легко управляти змінами в коді та співпрацювати з командою.

Live Share: Ця функція дозволяє розробникам спільно працювати над кодом в реальному часі, незалежно від того, де вони знаходяться, спрощуючи співпрацю та швидке вирішення проблем.

Профілювання та оптимізація додатків: Visual Studio надає інструменти для профілювання додатків, які допомагають ідентифікувати та вирішувати проблеми з продуктивністю та використанням ресурсів.

Ці та інші можливості роблять Visual Studio Enterprise 2022 відмінним середовищем для розробки мобільних додатків на Xamarin, надаючи розробникам потужні інструменти, які потрібні для створення високоякісних, ефективних та крос-платформних мобільних додатків.

Для розробки мобільних додатків на Xamarin через Visual Studio, треба встановити кілька ключових компонентів через Visual Studio Installer:

Xamarin SDKs - це набір інструментів, необхідних для розробки, компіляції та запуску мобільних додатків за допомогою Xamarin. Xamarin SDK включає бібліотеки, компілятори та рантайми, які дозволяють використовувати C# та .NET для створення додатків для Android, iOS та інших платформ.

Android SDK - це набір розробницьких інструментів від Google, який дозволяє розробляти додатки для Android. Включає компілятори, інструменти для налагодження, емулятори Android-пристроїв та доступ до останніх API Android. Xamarin розробникам потрібно встановити Android SDK, щоб створювати та тестувати додатки для Android.

iOS SDK. Аналогічно Android SDK, iOS SDK містить інструменти та API, необхідні для розробки додатків для iOS. Встановлення iOS SDK через Xcode на macOS є обов'язковим для розробки і налагодження додатків Xamarin.iOS, а також для використання інструментів дизайну інтерфейсу, таких як Interface Builder.

Universal Windows Platform (UWP) Tools. Якщо розробка ведеться також для Windows 10, то інструменти UWP дозволяють створювати додатки, які

можуть працювати на всіх пристроях під керуванням Windows 10, включаючи комп'ютери, планшети, телефони та інше.

Visual Studio Emulator for Android і iOS Simulator. Ці інструменти дозволяють розробникам емулювати пристрої Android та iOS безпосередньо на своєму робочому комп'ютері, що спрощує процес тестування та налагодження додатків.

Xamarin.Forms. Хоча Xamarin.Forms технічно є частиною Xamarin SDK, його можна виділити як окремий компонент через його роль у спрощенні розробки крос-платформних інтерфейсів користувача за допомогою XAML.

.NET MAUI (Multi-platform App UI). Якщо доступно, .NET MAUI є новітньою технологією, спрямованою на заміну Xamarin.Forms, для створення крос-платформних мобільних та десктопних додатків з єдиною кодовою базою.

Xamarin дозволяє розробникам використовувати C# для створення мобільних додатків, що працюють на кількох платформах, зокрема iOS, Android та Windows, з великою часткою спільного коду. Одна з основних переваг Xamarin - це ефективність роботи завдяки можливості поділу коду між платформами, що значно знижує час і витрати на розробку та тестування додатків. Крім того, використання C# дозволяє розробникам користуватися потужними функціями мови та .NET екосистеми, що забезпечує високий рівень продуктивності та безпеки.

Однак, хоча Xamarin і забезпечує велику сумісність коду між платформами, існують певні обмеження, які можуть вплинути на доступ до платформо-специфічних функцій та оптимізацію продуктивності. Розробники іноді можуть зіткнутися з труднощами при інтеграції специфічних для платформи бібліотек або при використанні новітніх функцій, які були недавно додані до нативних платформ. Це може призвести до додаткових витрат часу на розробку обгорток або пошук обхідних шляхів. Також, хоча продуктивність додатків Xamarin часто дуже близька до нативної, в деяких випадках можуть виникати проблеми з продуктивністю, особливо в ресурсоємних додатках.

Контрольні питання

1. Опишіть технологію Xamarin, її роль та місце у розробці крос-платформних мобільних додатків.
2. Опишіть архітектуру Xamarin.
3. Які можливості має Visual Studio Enterprise 2022 як інтегроване середовище розробки (IDE) для розробки мобільних додатків на Xamarin?
4. Опишіть основні компоненти Xamarin.
5. Опишіть переваги та недоліки Xamarin.

Порядок виконання лабораторної роботи

Налаштування Visual Studio Enterprise 2022

1. Панель управління -> Settigs -> Apps -> Visual Studio Enterprise 2022 -> Modify (Рис.1):

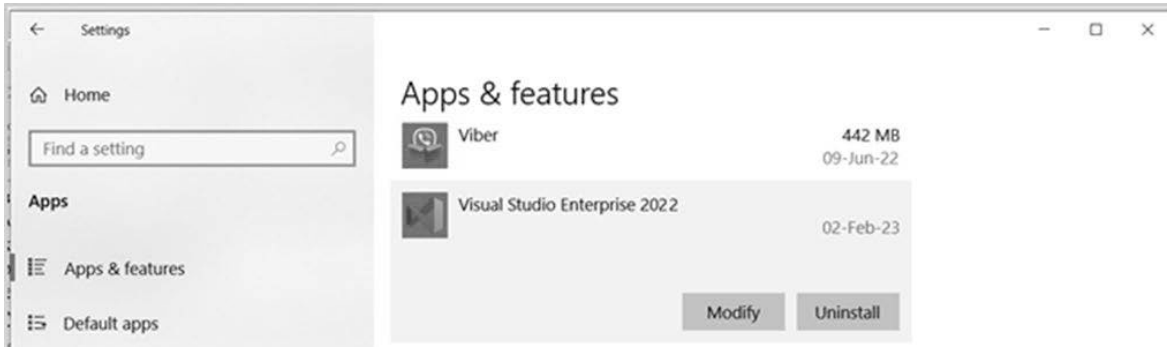


Рис.1.1. Налаштування Visual Studio Enterprise 2022

2. Вибрати Mobil development with.NET на вкладці Workloads (Рис.1.2):

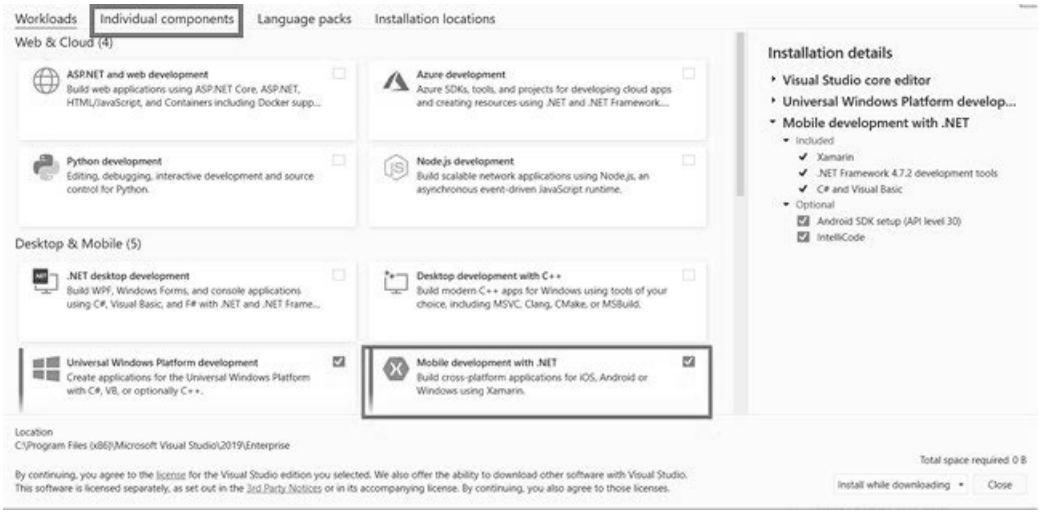


Рис. 1.2. Вкладка Workloads

3. Перейти на вкладку Individual components (Рис. 1.2) для перевірки налаштувань необхідних компонентів:

3.1. Development activities:

C# and Visual Basic, Xamarin, Xamarin Remoted Simulator (Рис.1.3):

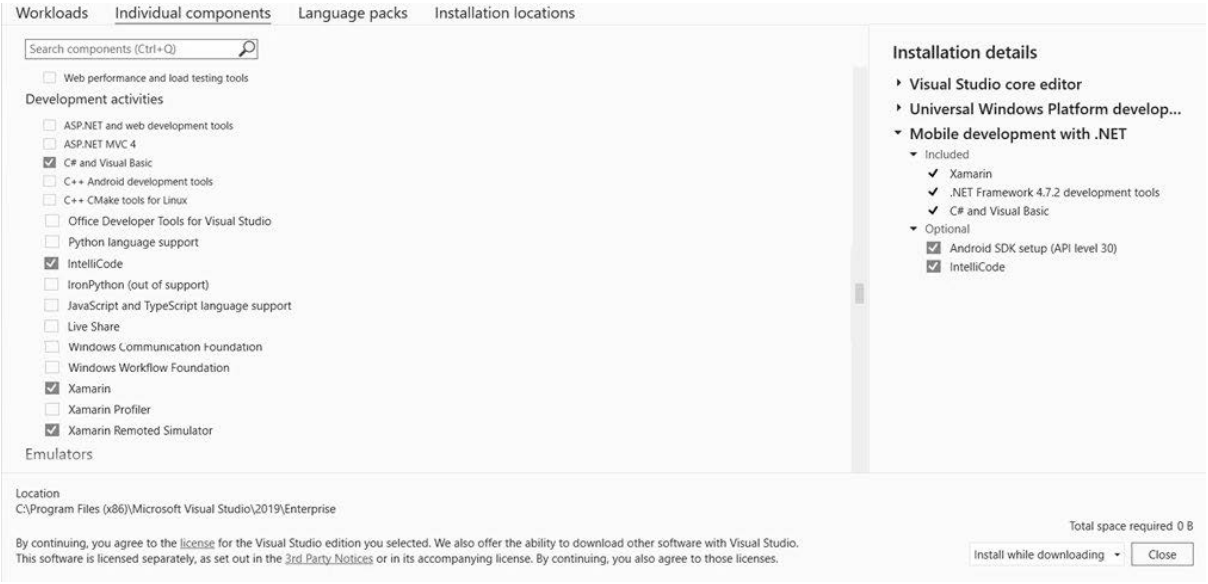


Рис. 1.3. Вкладка Individual components (Development activities)

3.2. SDKs, libraries, and frameworks: Android SDK setup (API level 30) (Рис. 1.4):

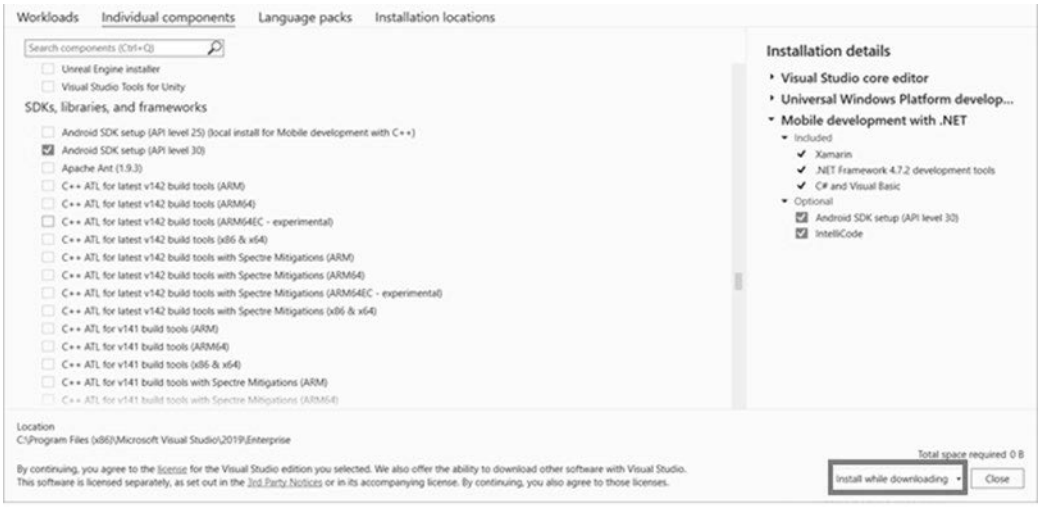


Рис.1.4. Вкладка Individual components (SDKs, libraries, and frameworks)

3.3. Emulators -> Google Android Emulator (API level 25) (local install), Intel Hardware Accelerated Exucution Manager (HAXM) (local install) (Рис. 1.5):

Рис. 1.5. Вкладка Individual components (продовження)

4. Натиснути Install while Downloading (Рис. 1.4)

Перевірка налаштувань Visual Studio:

5.1. Головне меню Tools -> Options -> Xamarin (Рис. 1.6):

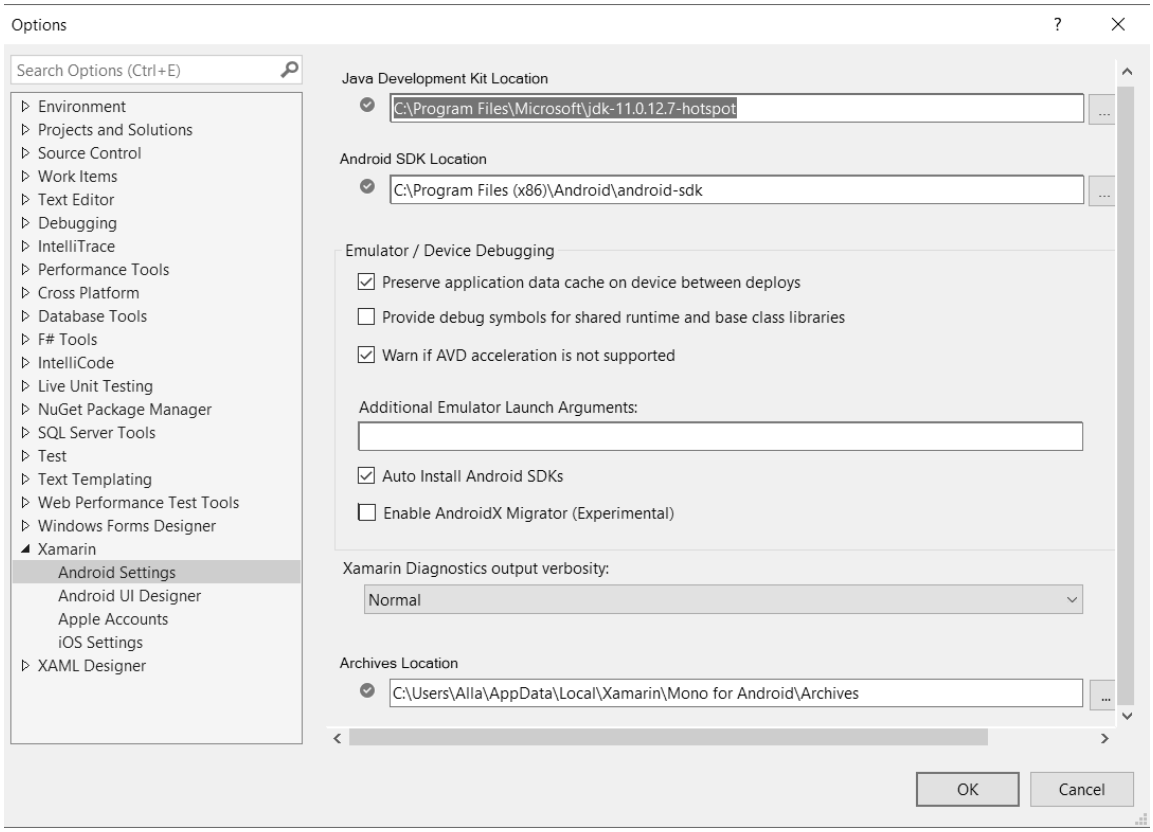
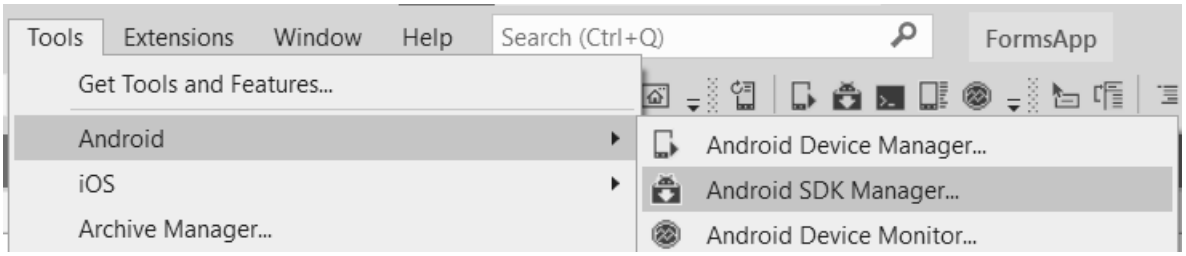


Рис. 1.6. Перевірка налаштувань Xamarin

5.2. Головне меню Tools -> Android -> Android SDK Manager…(Рис. 1.7, рис. 1.8):

До обов'язкових інструментів відноситься *Android SDK* (software development kit) - набір засобів програмування, який містить інструменти, необхідні для створення, компіляції та складання мобільного додатка, маючи компілятор, відладчик.



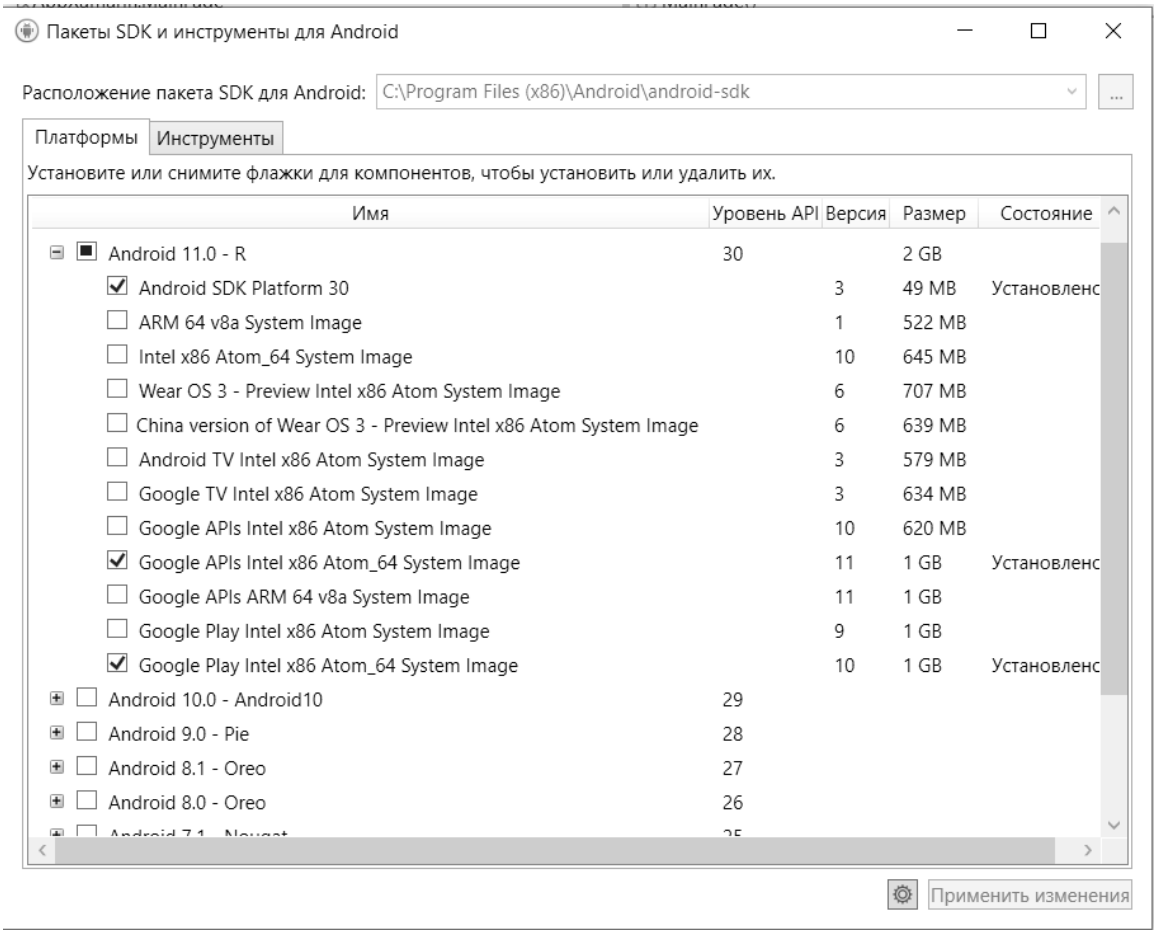


Рис. 1.7. Перевірка пакетів SDK для Android

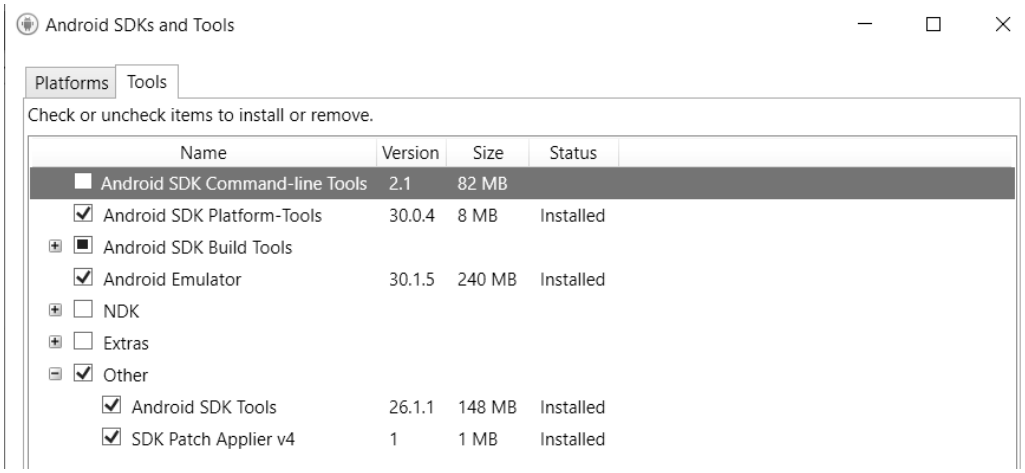


Рис. 1.8. Перевірка пакетів SDK для Android

6. У BIOS має бути включена віртуалізація (Intel Virtualization Technology)

Створення першої програми

1. Створити новий проект (Рис. 1.9):

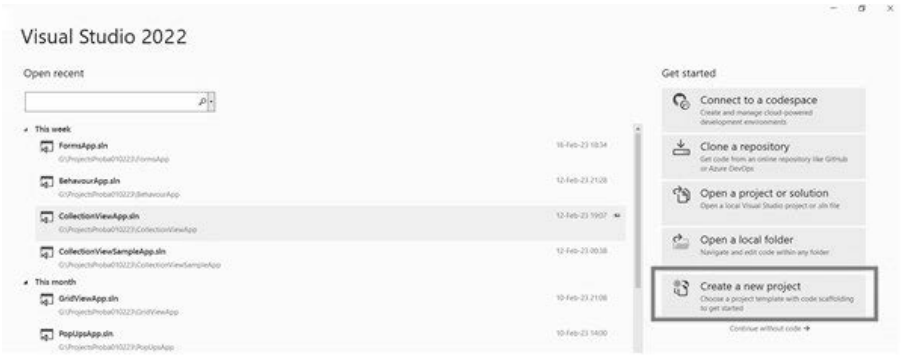


Рис. 1.9. Створення нового проекту

2. Виконайте пошук за словом "Xamarin" або виберіть Mobile у меню Project type. Виберіть тип проекту Mobile (Xamarin.Forms) (Рис. 1.10):

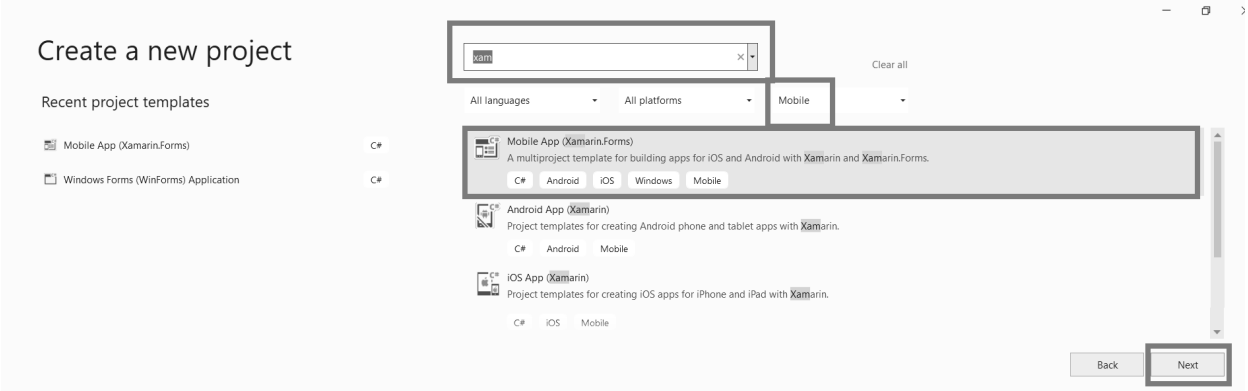


Рис. 1.10. Вибір типу проекту

3. Виберіть назву проекту (Рис. 1.11) — у прикладі використовується "HelloApp":

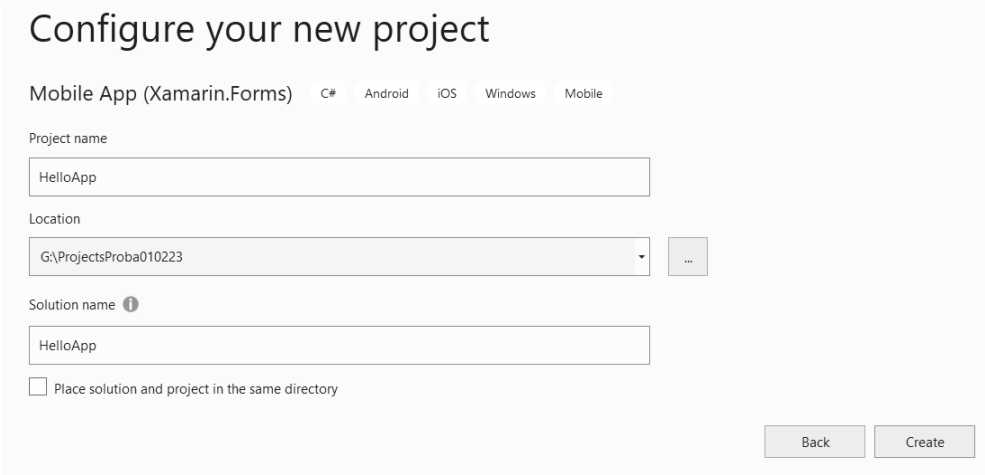


Рис. 1.11. Вибір назви проекту

4. Клацніть тип проекту Blank і переконайтеся, що вибрано параметр Android (Рис. 1.12):

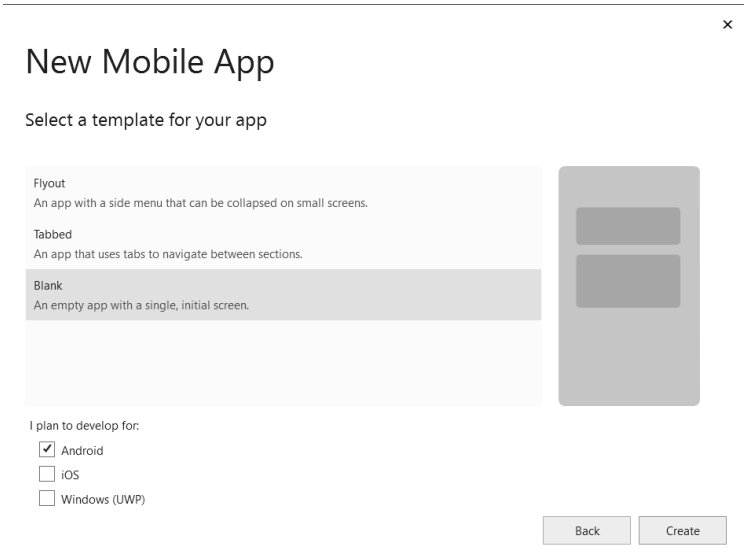


Рис. 1.12. Вибір операційної системи

5. Клацніть Create

6. Нові установки Visual Studio 2022 не містять налаштований емулятор Android. Клацніть стрілку списку, що розкривається, на кнопці Debug та виберіть пристрій (Рис. 1.13):

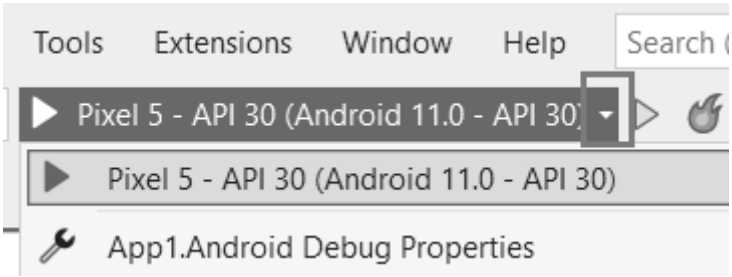
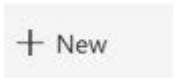


Рис. 1.13. Вибір пристрою

Або необхідно перейти на вкладку Tools -> Android -> Android Device Manager… вибрати пристрій та натиснути Start (Рис. 1.14):

Рис. 1.14. Вибір пристрою за допомогою пункту Tools головного меню

Якщо в Диспетчері пристроїв Android не згенерувався емулятор, то натиснути 

(Рис. 1.14) і створити новий пристрій.

7. У вікні, що з'явилося (Рис. 1.15) натисніть кнопку Create:

Рис. 1.15. Вікно створення Android Device

8. Для запуску проекту натисніть пункт Debug головного меню -> Start Debugging. З'явиться наступне вікно (Рис. 1.16):

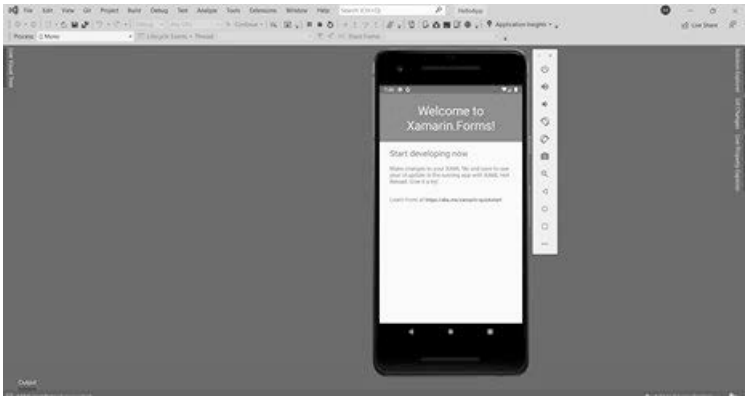


Рис. 1.16. Перевірка працездатності проекту

Підключення телефону

1. На телефоні вибрати: Установки - Система - Про телефон - Номер збирання. Номер збирання натиснути 7 разів.

2. Підключіть телефон до комп'ютера за допомогою шнура.

3. На телефоні з'явиться вікно «Режим роботи USB». Виберіть передачу файлів.

4. В Microsoft Visual Studio на панелі зі списку вибрати назву підключеного телефону:

5. Запустити додаток в Microsoft Visual Studio Debug –> Start Debugging. Додаток запуститься на телефоні (Рис. 1.17).

Рис. 1.17. Перевірка працездатності проекту на телефоні

Редагування програми

1. В Solution Explorer подвійним клацанням миші слід відкрити XAML-файл MainPage.xaml, який визначає головний екран програми.

Приклад 1.1.

```
<?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="HelloApp.MainPage">

    <StackLayout>
        <Frame BackgroundColor="#2196F3" Padding="24" CornerRadius="0">
            <Label Text="Welcome to Xamarin.Forms!"
HorizontalTextAlignment="Center" TextColor="White" FontSize="36"/>
        </Frame>
        <Label Text="Start developing now" FontSize="Title"
Padding="30,10,30,10"/>
        <Label Text="Make changes to your XAML file and save to see your
UI update in the running app with XAML Hot Reload. Give it a try!"
FontSize="16" Padding="30,0,30,0"/>
        <Label FontSize="16" Padding="30,24,30,0">
            <Label.FormattedText>
                <FormattedString>
                    <FormattedString.Spans>
                        <Span Text="Learn more at "/>
                        <Span Text="https://aka.ms/xamarin-quickstart"
FontAttributes="Bold"/>
                    </FormattedString.Spans>
                </FormattedString>
            </Label.FormattedText>
        </Label>
    </StackLayout>
</ContentPage>
```

2. Відредагуйте вміст вихідного файлу наступним чином, записавши до другого елемента Label своє прізвище, а до третього елемента Label – свою групу.

Приклад 1.2.

```
<?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="HelloApp.MainPage">
    <StackLayout>
        <Frame BackgroundColor="#2196F3" Padding="24" CornerRadius="0">
            <Label Text="Hello!" HorizontalTextAlignment="Center"
TextColor="White" FontSize="36"/>
        </Frame>
```

```
        <Label Text="My name is White" FontSize="Title"
Padding="30,10,30,10"/>
        <Label Text="My group is 1" FontSize="16" Padding="30,0,30,0"/>
     </StackLayout>
</ContentPage>
```

Завдання

1. Налаштуйте Visual Studio для створення мобільних програм засобами Xamarin.

2. Підключіть емулятор мобільного телефону до Visual Studio.

3. Підключіть мобільний телефон до Visual Studio.

4. Створіть мобільний додаток, який виводить на екран «Hello!», Ваше прізвище, ім'я та групу.

5. Запустіть програму на емуляторі та телефоні (Рис. 1.18).

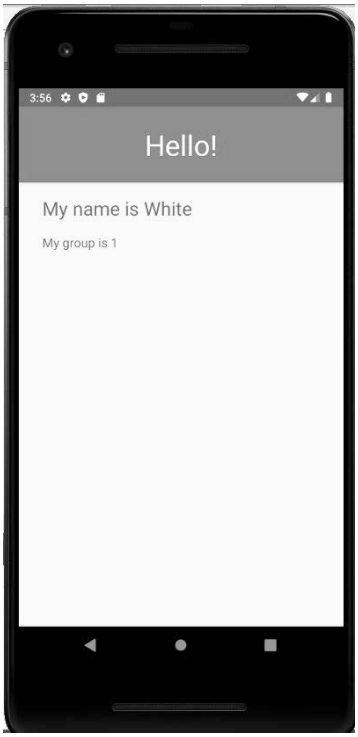


Рис. 1.18. Результат роботи проекту

ЛАБОРАТОРНА РОБОТА №2
СТВОРЕННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА ДОДАТКУ. ЕЛЕМЕНТ КОМПОНУВАННЯ STACKLAYOUT

Мета роботи: оволодіти навичками створення інтерфейсу користувача для мобільного додатку за допомогою Xamarin.Forms, навчитися працювати з основними елементами управління та компонуванням вікна додатку.

Теоретичні відомості

За своєю архітектурою програми Xamarin.Forms не відрізняються від традиційних кросплатформних додатків. Загальний код зазвичай розміщується в бібліотеці .NET Standard і використовується програмами для конкретних платформ.

При виконанні Xamarin.Forms-програми відбувається таке:

- Код мовою C# компілюється в байт-код Microsoft Intermediate Language (MSIL), який може виконуватися на будь-якій платформі, що підтримує .NET.
- Під час запуску програми на конкретній платформі (наприклад, Android або iOS) відбувається завантаження бібліотек .NET Runtime та Xamarin.Forms на пристрій.
- Байт-код MSIL переводиться в нативний код, сумісний з апаратною архітектурою пристрою, на якому запускається програма, за допомогою JIT (Just-In-Time) компіляції.
- Потім, залежно від конкретної платформи, створюється і відображається інтерфейс користувача (UI), використовуючи нативні елементи інтерфейсу.
- Коли користувач взаємодіє з додатком, Xamarin.Forms перехоплює події та викликає відповідні методи у C#-коді.
- У свою чергу, C#-код може звертатися до нативних API та виконувати будь-які операції, що підтримуються платформою.
- Коли програма завершує роботу, нативний код вивантажується з пам'яті пристрою, а сесія .NET Runtime завершується.

З точки зору логіки програми та інтерфейсу, виконання програми починається з файлів App.xaml та App.xaml.cs.

У файлі App.xaml.cs. є клас App, який відповідає за створення екземпляра програми на кожній платформі.

Приклад 2.1.

```
public partial class App : Application
  {
    public App()
    {
      InitializeComponent();
      MainPage = new MainPage();
     }
    }
```

Цей файл встановлює головну сторінку програми через властивість MainPage у конструкторі. У C# ключове слово partial використовується для створення класу, який можна розділити на кілька файлів.

Приклад 2.2. 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="AppXamarin.App">
  <Application.Resources>

  </Application.Resources>
</Application>
```

Визначення головної сторінки розбито на два файли.

MainPage.xaml - це візуальний інтерфейс сторінки у вигляді коду XAML.

Приклад 2.3.

```
<?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="Lab2App.MainPage">
    <StackLayout>
        <Frame BackgroundColor="#2196F3" Padding="24" CornerRadius="0">
            <Label Text="Welcome!" HorizontalTextAlignment="Center"
TextColor="White" FontSize="36"/>
        </Frame>
    </StackLayout>
</ContentPage>
```

Так як для створення сторінок використовується клас ContentPage, то як кореневий елемент у цьому файлі визначено саме елемент ContentPage.

Також у проекті є і файл із кодом логіки сторінки - файл MainPage.xaml.cs.

Приклад 2.4.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace Lab2App
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
 }
```

Через виклик у конструкторі InitializeComponent() на сторінці формується інтерфейс, визначений у файлі MainPage.xaml.

Щоб зв'язати обидва визначення сторінки, клас MainPage визначається як частковий (partial), а у файлі MainPage.xaml у визначенні сторінки ContentPage зазначений клас, назва якого збігається з класом з MainPage.xaml.cs.

Приклад 2.5.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class=" Lab2App.MainPage">
```

Xamarin.Forms дозволяє створювати єдиний код для програм на різних платформах (Android, iOS, UWP), використовуючи різні елементи інтерфейсу та механізми навігації, які максимально відповідають стандартам кожної платформи. Він також надає велику кількість елементів інтерфейсу користувача, таких як кнопки, текстові поля, зображення, таблиці і багато інших, які можна використовувати на всіх платформах.

Користувальницький інтерфейс

Для створення інтерфейсу користувача Xamarin.Forms в додатках використовуються кілька груп елементів управління:

Сторінки - сторінки Xamarin.Forms представляють екрани в кросплатформових мобільних додатках. Для відображення окремих екранів зазвичай використовується клас ContentPage.

Представлення - представлення Xamarin.Forms є елементами керування, що відображаються в інтерфейсі користувача, наприклад, мітки, кнопки і поля введення тексту.

Макети - макети Xamarin.Forms є контейнерами, які служать для об'єднання представлень в логічні структури. У програмах часто використовується клас StackLayout для компонування уявлень у стеку, а також клас Grid для компонування кнопок по горизонталі.

У Xamarin.Forms візуальний інтерфейс складається зі сторінок. Сторінка є об'єктом класу Page, вона займає весь простір екрану. Тобто те, що користувач бачить на екрані мобільного пристрою – це сторінка. Програма може мати одну або кілька сторінок.

Сторінка як вміст приймає один із контейнерів компонування, в який у свою чергу поміщаються стандартні візуальні елементи типу кнопок та текстових полів, а також інші елементи компонування.

Елементи компонування

У Xamarin можна використовувати низку елементів компонування. Їх поєднує те, що вони успадковані від загального класу View і тому успадковують ряд загальних властивостей.

Крім звичайних елементів типу кнопок і текстових полів, в Xamarin.Forms також є контейнери, які дозволяють скомпонувати вміст, розташувати його певним чином.

Для визначення вмісту сторінки клас сторінки ContentPage має властивість Content. За умовчанням цій властивості присвоюється один елемент Label.

Властивість Content має обмеження – для неї можна встановити лише один елемент. І щоб поміщати на сторінку відразу кілька елементів, треба використовувати один із елементів компонування. Елемент компонування є класом, який успадковується від базового класу Layout<T>:

- StackLayout;
- AbsoluteLayout;
- RelativeLayout;
- Grid;
- FlexLayout.

Xamarin.Forms дозволяє створювати візуальний інтерфейс як за допомогою C# коду, так і декларативним шляхом за допомогою мови XAML, або комбінуючи ці підходи.

StackLayout

StackLayout визначає розміщення елементів як горизонтального чи вертикального стека. Для позиціонування елементів визначено властивості:

– Orientation: визначає орієнтацію стека – вертикальний чи горизонтальний.

Завдання властивості для C#:

Orientation = StackOrientation.Horizontal

та для XAML:

StackLayout.Orientation = "Horizontal"

– Spacing: встановлює простір між елементами у стеку, за замовчуванням дорівнює 6 одиницям.

За умовчанням в Xamarin використовується одиниця вимірювання "діпі" (dp), яка є відносною одиницею вимірювання, що враховує щільність пікселів на екрані пристрою.

Це дозволяє створювати адаптивні інтерфейси, які можуть автоматично масштабуватися на різних пристроях з різними щільностями пікселів.

Всі елементи компонування мають властивість Children, що дозволяє встановити або отримати вкладені елементи.

Елемент StackLayout - звичайний елемент, який може використовувати всі стандартні властивості типу налаштування кольору, відступи і т.д.

Клас StackLayout спрощує розробку кросплатформових програм завдяки автоматичному компонування уявлень на екрані незалежно від його розміру. Дочірні елементи розміщуються по черзі (по горизонталі або по вертикалі) у порядку додавання. Область, яку займає макет StackLayout, залежить від значень властивостей HorizontalOptions і VerticalOptions, але за замовчуванням макет StackLayout намагатиметься використовувати весь екран.

Приклад 2.6. Встановлення вкладених елементів Label у коді C# (файл MainPage.xaml.cs):

```
public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            Label label1 = new Label()
            {
                Text = "My name is White",
```

```
            TextColor = Color.Red
        };
        Label label2 = new Label()
        {
            Text = " My group is 1",
            TextColor = Color.Blue
        };

        StackLayout stackLayout = new StackLayout()
        {
            Children = { label1, label2 },
            Spacing = 8
        };

        this.Content = stackLayout;

    }

}
```

Відображення вкладених елементів Label на екрані наведено на Рис. 2.1.

Рис. 2.1. Відображення вкладених елементів на екрані

Аналогічно у XAML можна написати (файл MainPage.xaml).

Приклад 2.7.

```
<?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="Lab2App.MainPage">

    <StackLayout x:Name="stackLayout" Spacing="8">
        <StackLayout.Children>
            <Label Text="My name is White" TextColor="Red"  />
            <Label Text="My group is 1" TextColor="Blue" />
        </StackLayout.Children>
    </StackLayout>
</ContentPage>
```

або:

```
<?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="Lab2App.MainPage">

    <StackLayout x:Name="stackLayout" Spacing="8">
        <Label Text="My name is White" TextColor="Red" />
        <Label Text="My group is 1" TextColor="Blue" />
    </StackLayout>

</ContentPage>
```

BoxView

Елемент BoxView представляє прямокутник. Найчастіше BoxView використовується для створення пофарбованих областей або як декоративне примітивне графічне оформлення інших елементів.

Основні властивості класу BoxView:

– Color: представляє колір елемента як структури Color.

– CornerRadius: представляє радіус межі BoxView у вигляді значення типу float.

– WidthRequest: представляє ширину елемента (за замовчуванням дорівнює 40 одиниць).

– HeightRequest: представляє висоту елемента (за замовчуванням дорівнює 40 одиниць).

– HorizontalOptions і VerticalOptions: ці властивості керують тим, як BoxView вирівнюється всередині батьківського контейнера, приймають значення, задані в структурі LayoutOptions.

HeightRequest та WidthRequest: ці властивості визначають бажані розміри BoxView. Якщо не вказано, BoxView буде займати стільки місця, скільки потрібно для відображення вмісту.

Структура LayoutOptions Xamarin.Forms визначає опції розміщення елементів управління всередині контейнера. LayoutOptions використовується в різних елементах управління Xamarin.Forms, таких як Grid або StackLayout, щоб визначити, як елементи повинні бути розміщені всередині контейнера. Вона також може використовуватися для налаштування опцій розміщення елементів у компонентах користувача.

Вона містить такі властивості:

Start - вирівнювання елемента управління по лівому/верхньому краю контейнера.

Center - вирівнювання елемента керування по центру контейнера.

End - вирівнювання елемента управління з правого/нижнього краю контейнера.

Fill - елемент керування заповнює всю доступну горизонтальну / вертикальну область контейнера.

Коли встановлено властивість LayoutOptions дочірнього елемента FillAndExpand, елемент буде розтягнутий по горизонталі і вертикалі настільки, наскільки це можливо, щоб зайняти все вільне місце всередині батьківського контейнера.

Значення LayoutOptions, такі як CenterAndExpand та EndAndExpand, використовуються для вирівнювання елементів по центру або краю батьківського контейнера, але вони не заповнюють все доступне місце всередині контейнера.

Створення сторінки (Рис. 2.2) з двома прямокутниками (червоним та синім) наведено в прикладі 2.8. Використовуються елементи BoxView та StackLayout.

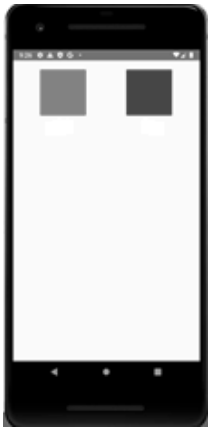


Рис. 2.2. Сторінка із двома прямокутниками

Приклад 2.8. На C# (файл MainPage.xaml.cs):

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace Lab2App
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
```

```
            BoxView boxView1 = new BoxView
            {
                Color = Color.FromRgb(255, 0, 0),
                WidthRequest = 100,
                HeightRequest = 100,
                HorizontalOptions = LayoutOptions.CenterAndExpand,
                VerticalOptions = LayoutOptions.Center
            };

            BoxView boxView2 = new BoxView
            {
                Color = Color.Blue,
                WidthRequest = 100,
                HeightRequest = 100,
                HorizontalOptions = LayoutOptions.CenterAndExpand,
                VerticalOptions=LayoutOptions.Center
            };

            StackLayout stackLayoutH1 = new StackLayout()
            {
                Children = { boxView1,boxView2 },
                Spacing = 8,
                Margin=20,
                Orientation = StackOrientation.Horizontal
            };

            this.Content = stackLayoutH1;
        }
    }
}
```

Приклад 2.9. На XAML (файл MainPage.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="Lab2App.MainPage">
    <StackLayout  Margin="20" Spacing="8"  StackLayout.Orientation
="Horizontal">
        <BoxView    Color="Red"   HorizontalOptions="CenterAndExpand"
VerticalOptions="Center" WidthRequest ="100" HeightRequest ="100"/>
        <BoxView   Color="Blue" HorizontalOptions ="CenterAndExpand"
VerticalOptions="Center" WidthRequest ="100" HeightRequest ="100"/>
    </StackLayout>
  </ContentPage>
```

Контрольні питання

1. Які події відбуваються при виконанні Xamarin.Forms-програми?
2. На які два файли розбито визначення головної сторінки?
3. Які групи елементів управління використовуються в додатках для створення інтерфейсу користувача?
4. Основні властивості елементу компонування StackLayout.
5. Основні властивості та призначення класу BoxView.

Завдання

Створити сторінку з 9 різнобарвними прямокутниками (по 3 прямокутники в кожному рядку) двома засобами C# і XAML. Під кожним прямокутником на сторінці вказати колір. Використовувати елементи BoxView, Label, та StackLayout.

ЛАБОРАТОРНА РОБОТА №3
ВИВЧЕННЯ ЕЛЕМЕНТІВ ІНТЕРФЕЙСУ

Мета роботи: оволодіти навичками створення інтерфейсу користувача для мобільного додатку за допомогою Xamarin.Forms, навчитися працювати з основними елементами управління та компонуванням вікна додатку.

Теоретичні відомості

Текстові поля

У Xamarin Forms текстові поля представлені кількома класами:

- LABEL: текстова мітка для виведення тексту;
- ENTRY: однорядкове текстове поле;
- EDITOR: багаторядкове текстове поле.

Label

Label представляє звичайну текстову мітку, яка виводить інформацію за допомогою властивості Text.

Текстове поле Entry

Entry представляє текстове поле для введення однорядкової інформації.

Основні властивості елемента Entry:

- Text: введений у полі текст;
- TextColor: колір тексту, що вводиться;
- Placeholder: текст-замінник, який відображається в полі до введення та зникає на початку друку;
- IsPassword: при значенні True вказує, що поле буде призначене для введення пароля, і тому всі символи, що вводяться, будуть замінюватися зірочкою.

Також клас Entry визначає події:

- TextChanged: виникає при введенні символів у поле;
- Completed: виникає після завершення введення;
- Keyboard: формат клавіатури.

Формат клавіатури

Серед властивостей текстового поля введення найцікавішою є властивість Keyboard. Вона дозволяє встановити формат клавіатури для введення символів. Наприклад, якщо необхідно ввести в поле якусь цифрову інформацію, то було б простіше з точки зору користувача надати користувачеві відразу числову розкладку клавіатури.

Ця властивість приймає одне із значень перерахування Keyboard:

- Default;
- Text;

– Chat;
– Url;
– Email;
– Telephone;
– Numeric.

Кожне значення надає розкладку клавіатури, призначену для введення певної інформації. Наприклад, entry.Keyboard=Keyboard.Telephone - при введенні автоматично подається розкладка, що містить лише ті знаки, які використовуються при наборі телефонного номера.

Editor

На відміну від Entry Editor є багаторядковим полем введення, але принципи його роботи аналогічні. Він також має ті самі властивості і події.

Приклад використання елементів Entry, Label. Введені користувачем дані до елементів Entry виводяться в елементи Label.

Приклад 3.1. Реалізація інтерфейсу програми (рис.3.1) у XAML:

```
<StackLayout>
    <Entry x:Name="parameter1Entry" Placeholder = "Enter parameter1"
TextChanged="parameter1Entry_TextChanged" Keyboard="Numeric"/>
    <Entry x:Name="parameter2Entry" Placeholder = "Enter parameter2"
TextChanged="parameter2Entry_TextChanged" />
    <Label x:Name="textLabel1" FontSize="Large" />
    <Label x:Name="textLabel2" />
</StackLayout>
```

Рис. 3.1. Інтерфейс користувача

Приклад 3.2. Реалізація логіки програми у C#:

```
public partial class MainPage : ContentPage
    {
       public MainPage()
       {
           InitializeComponent();
       }
       private void parameter1Entry_TextChanged(object sender,
TextChangedEventArgs e)
       {
           Entry parameter1Entry = (Entry)sender;
           textLabel1.Text = parameter1Entry.Text;
       }
       private void parameter2Entry_TextChanged(object sender,
TextChangedEventArgs e)
       {
           Entry parameter2Entry = (Entry)sender;
           textLabel2.Text = parameter2Entry.Text;
       }
```

Метод parameter1Entry_TextChanged є обробником подій для події TextChanged, яка виникає, коли текст у полі введення (Entry) змінюється.

Метод parameter1Entry_TextChanged:

1. Метод приймає два параметри: object sender і TextChangedEventArgs e:

– sender є об'єктом, який викликав подію. У цьому випадку, це поле введення parameter1Entry.

– e містить дані події, включно з попереднім і новим значеннями тексту.

2. Першою дією в методі є приведення sender до типу Entry, що дозволяє доступатися до властивостей і методів поля введення:

Entry parameter1Entry = (Entry)sender;

Це робиться для того, щоб можна було використовувати властивість Text цього поля введення.

3. Наступним кроком є присвоєння тексту з поля введення (parameter1Entry.Text) до текстового поля (textLabel1.Text). Це означає, що коли користувач вводить текст у поле введення, текст миттєво відображається в Label з іменем textLabel1.

Приклад 3.3. Реалізація інтерфейсу і логіки у C#:

```
public partial class MainPage : ContentPage
    {
        Label textLabel1, textLabel2;
        Entry parameter1Entry, parameter2Entry;
        public MainPage()
        {
```

```
            InitializeComponent();
            StackLayout stackLayout = new StackLayout();

            parameter1Entry = new Entry { Placeholder = "Enter
parameter1", Keyboard=Keyboard.Numeric };
            parameter1Entry.TextChanged += parameter1Entry_TextChanged;
            textLabel1 = new Label { FontSize =
Device.GetNamedSize(NamedSize.Large, typeof(Label)) };
            parameter2Entry = new Entry { Placeholder = "Enter
parameter2" };
            parameter2Entry.TextChanged += parameter2Entry_TextChanged;
            textLabel2 = new Label ();

            stackLayout.Children.Add(parameter1Entry);
            stackLayout.Children.Add(parameter2Entry);
            stackLayout.Children.Add(textLabel1);
            stackLayout.Children.Add(textLabel2);
            this.Content = stackLayout;

        }

        private void parameter1Entry_TextChanged(object sender,
TextChangedEventArgs e)
        {
            textLabel1.Text = parameter1Entry.Text;
        }

        private void parameter2Entry_TextChanged(object sender,
TextChangedEventArgs e)
        {
            textLabel2.Text = parameter2Entry.Text;
        }
    }
```

Кнопки

Для створення будь-якої дії знадобляться кнопки, представлені класом Button. Наприклад, при натисканні користувачем на кнопку «PRESS ME!», кнопка змінює колір та напис.

Приклад 3.4. Реалізація інтерфейсу програми для рис. 3.2 в XAML.

```
<StackLayout>
        <Button Text = "Press me!" FontSize="Large" BorderWidth="1"
        HorizontalOptions="Center" VerticalOptions="CenterAndExpand"
        Clicked="OnButtonClicked" />
</StackLayout>
```

Рис. 3.2. Приклад використання кнопки.

У файлі відокремленого коду MainPage.xaml.cs треба прописати обробник OnButtonClicked.

Приклад 3.5. Обробник OnButtonClicked

```
public partial class MainPage : ContentPage
    {
           public MainPage()
            {
            InitializeComponent();
            }
        private void OnButtonClicked(object sender, System.EventArgs e)
        {
            Button button = (Button)sender;
            button.Text = "Pressed!";
            button.BackgroundColor = Color.Red;
        }
    }
```

Приклад 3.6. Реалізація інтерфейсу і логіки на C#.

```
public MainPage()
            {
                StackLayout stackLayout = new StackLayout();
                Button button = new Button
                {
                    Text = "Press me!",
                    FontSize = Device.GetNamedSize(NamedSize.Large,
typeof(Button)),
                    BorderWidth = 1,
                    HorizontalOptions = LayoutOptions.Center,
                    VerticalOptions = LayoutOptions.CenterAndExpand
                };
                button.Clicked += OnButtonClicked;
                stackLayout.Children.Add(button);
                this.Content = stackLayout;
            }
```

```
private void OnButtonClicked(object sender, System.EventArgs e)
  {
      Button button = (Button)sender;
      button.Text = "Pressed!";
      button.BackgroundColor = Color.Red;
  }
```

Device.GetNamedSize(NamedSize.Large, typeof(Button)) дозволяє отримати конкретний розмір шрифту, який Xamarin.Forms вважає "великим" для елементів типу Button на різних платформах і пристроях.

Xamarin.Forms має декілька іменованих розмірів шрифтів (Micro, Small, Medium, Large, Default), і реальний розмір шрифту, який вони представляють, може варіюватися в залежності від платформи (iOS, Android, UWP) і налаштувань пристрою. Ця функція забезпечує більшу уніфікацію та адаптивність дизайну інтерфейсу, дозволяючи розробникам встановлювати "зручний для читання" розмір шрифту без необхідності жорсткого кодування конкретних значень пікселів або точок.

Для кнопки задано обробник натискання для події Clicked OnButtonClicked. У цьому випадку після натискання просто змінюється текст і колір фону кнопки.

Обробник багато в чому аналогічний стандартним обробникам у Windows Forms. Він приймає два параметри: об'єкт типу object (джерела події) та System.EventArgs (аргумент події, що зберігає деяку додаткову інформацію).

Контрольні питання

1. Які основні класи представлені в Xamarin Forms для роботи з текстовими полями?
2. Що робить властивість IsPassword у класі Entry?
3. Які події визначає клас Entry?
4. Для чого використовується властивість Keyboard в текстовому полі введення?
5. Чим відрізняється Entry від Editor в Xamarin Forms?
6. Як можна отримати "великий" розмір шрифту для кнопки у Xamarin Forms, і чому це значення може відрізнятися на різних платформах?

Завдання

1. Розробіть інтерфейс для калькулятора (рис. 3.3) за допомогою елемента компонування StackLayout мовою XAML.

2. Додайте обробники подій для кнопок та реалізуйте логіку калькулятора за допомогою C#.

3. Розробіть інтерфейс для калькулятора (рис. 3.3) за допомогою C# і реалізуйте логіку калькулятора на C#.

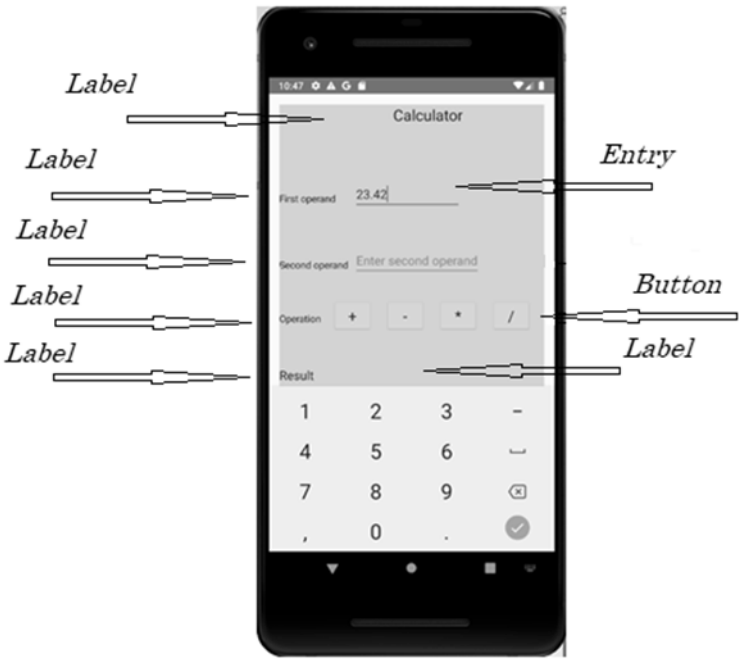


Рис. 3.3. Інтерфейс для калькулятора.

ЛАБОРАТОРНА РОБОТА №4
ВИВЧЕННЯ КОНТЕЙНЕРІВ КОМПОНУВАННЯ. ЕЛЕМЕНТ RELATIVELAYOUT

Мета: вивчення контейнеру компонування RelativeLayout для створення гнучких і адаптивних інтерфейсів користувача, які коректно відображаються на пристроях з різними розмірами екранів.

Теоретичні відомості

Контейнер RelativeLayout визначає відносне позиціонування вкладених елементів щодо сторін контейнера або щодо інших елементів.

Розміри елементів

При установці розмірів елементів у RelativeLayout у XAML можна використовувати дві стратегії:

- встановити значення через властивості HeightRequest та WidthRequest у елементів;
- встановити обмеження RelativeLayout.HeightConstraint та RelativeLayout.WidthConstraint.

Використання HeightRequest та WidthRequest

Ці властивості вказують запитувані (бажані) висоту та ширину для елемента. Вони просто пропонують RelativeLayout конкретний розмір елементу. Однак, фактичний розмір елемента може залежати від доступного простору, обмежень контейнера та інших елементів у макеті. Це досить простий спосіб задати розмір, але він не завжди гарантує, що елемент матиме саме такий розмір, особливо в складних макетах.

Використання RelativeLayout.HeightConstraint та RelativeLayout.WidthConstraint

Ці обмеження (constraints) дозволяють вказати більш гнучкі правила для визначення розмірів елементів, засновані на властивостях інших елементів або самого контейнера RelativeLayout. Можна встановити розмір елемента як відсоток від розміру батьківського контейнера, або відносно розмірів інших елементів. Це дозволяє створювати динамічні та адаптивні інтерфейси, які краще пристосовуються до різних розмірів екрану та орієнтацій.

Основна відмінність між цими двома підходами полягає в гнучкості та контролі над розміром елементів. Використання HeightRequest та WidthRequest пропонує простий спосіб задати бажані розміри, тоді як обмеження (constraints) RelativeLayout надають більше можливостей для створення респонсивного дизайну, що адаптується до змін у розмірах контейнера.

Приклад 4.1. Використання в BoxView HeightRequest та WidthRequest.

```
<RelativeLayout>
    <BoxView WidthRequest="100" HeightRequest="100" Color="Blue"
      RelativeLayout.XConstraint ="{ConstraintExpression Type=Constant, Constant=0}"
      RelativeLayout.YConstraint="{ConstraintExpression Type=Constant, Constant=0}"      />
  </RelativeLayout>
```

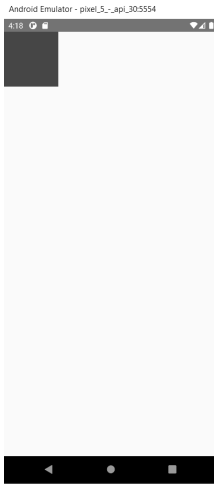


Рис. 4.1. Інтерфейс для прикладу 4.1.

Приклад 4.2. Попередній приклад на C#:

```
public MainPage()
        {
                InitializeComponent();
      RelativeLayout relativeLayout = new RelativeLayout();

      BoxView blueBox = new BoxView { BackgroundColor = Color.Blue, HeightRequest = 100,
WidthRequest = 100 };

      relativeLayout.Children.Add(blueBox,
         Constraint.Constant(0),
         Constraint.Constant(0)
      );
      Content = relativeLayout;
         }
```

Метод Children.Add() у RelativeLayout має ряд перевантажень, що дозволяє визначити відносні координати елемента. Як перший параметр передається сам елемент. Інші параметри не є обов'язковими, і їх набір можна варіювати. Він послідовно приймає значення для x-координати, y-координати, ширини та висоти. Другий і третій параметри методу встановлюють координати X і Y верхнього лівого кута елемента, які можуть задаватися методом Constraint.Constant.

Те ж саме з використанням RelativeLayout.HeightConstraint та RelativeLayout.WidthConstraint. Розміри прямокутника є 1/3 від розмірів RelativeLayout.

Приклад 4.3.

```
<RelativeLayout>
    <BoxView Color="black"
      RelativeLayout.XConstraint ="{ConstraintExpression Type=Constant, Constant=0}"
      RelativeLayout.YConstraint="{ConstraintExpression Type=Constant, Constant=0}"
      RelativeLayout.HeightConstraint="{ConstraintExpression Type=RelativeToParent,
Property=Height, Factor=0.33}"
      RelativeLayout.WidthConstraint    ="{ConstraintExpression Type=RelativeToParent,
Property=Width, Factor=0.33}"

    />
</RelativeLayout>
```

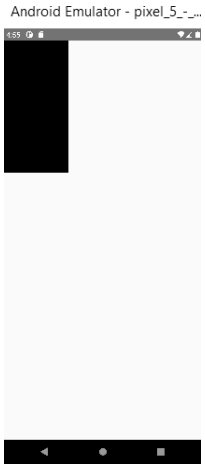


Рис. 4.3. Інтерфейс для прикладу 4.3.

Те ж саме на C#.

Приклад 4.4.

```
public MainPage()
    {
      InitializeComponent();
      RelativeLayout relativeLayout = new RelativeLayout();

      BoxView blueBox = new BoxView { BackgroundColor = Color.Black };

      relativeLayout.Children.Add(blueBox,
          Constraint.Constant(0),
          Constraint.Constant(0),
        Constraint.RelativeToParent((parent) =>
        {
           return parent.Width * 0.33;
```

```
    }),
     Constraint.RelativeToParent((parent) =>
     {
        return parent.Height * 0.33;
     })
                        );
Content = relativeLayout;
  }
```

Вираз (parent) => { return parent.Width *0.33; } використовується в програмуванні для створення анонімної функції, яка приймає один аргумент — в даному випадку, parent. Цей аргумент представляє батьківський елемент, до якого застосовується функція. В контексті Xamarin.Forms та його RelativeLayout, parent зазвичай є батьківським контейнером, у якому розміщується елемент.

=> називається "лямбда оператором" і використовується для визначення анонімної функції або лямбда-виразу. Він ділить лямбда-вираз на дві частини: зліва від оператора знаходяться параметри функції (у цьому випадку це parent), а справа — тіло функції, яке описує дії, що виконуються функцією. У даному випадку, тіло функції return parent.Width *0.33; повертає значення ширини батьківського елемента *0.33 пікселів.

Лямбда-вираз (parent) => { return parent.Width *0.33; }дозволяє динамічно обчислювати значення на основі стану іншого об'єкта, дозволяє Xamarin.Forms викликати цю функцію кожен раз, коли потрібно визначити позицію елемента, дозволяючи адаптуватися до змін у розмірах батьківського елемента. Лямбда-вирази дозволяють визначати функції "на льоту", які можуть бути передані як аргументи іншим методам або збережені в змінних для пізнішого використання.

Позиціювання щодо іншого елемента

Позиціонування всередині RelativeLayout визначається за допомогою обмежень, які в XAML представляють такі властивості, що прикріплюються:

- RelativeLayout.XConstraint: задає розташування щодо осі X;
- RelativeLayout.YConstraint: задає розташування щодо осі Y.

RelativeLayout.XConstraint та RelativeLayout.YConstraint задаються за допомогою розширення розмітки ConstraintExpression, яке включає таку інформацію:

- Type: тип обмеження, що вказує, застосовується обмеження щодо контейнера чи інших елементів;
- Property: властивість, на основі якої встановлюється обмеження;
- Factor: множник, який множиться довжина між межами контейнера (0 і 1 - крайні значення);

- Constant: зміщення щодо контейнера чи щодо елемента (залежно від значення властивості Type);

- ElementName: назва елемента, до якого застосовується обмеження.

При позиціонуванні щодо іншого елемента необхідно встановити параметр ElementName.

Приклад позиціонування щодо елемента у XAML.

Приклад 4.5.

```
<RelativeLayout>
        <Label x:Name="lbl" Text="RelativeLayout"
             RelativeLayout.XConstraint = "{ConstraintExpression
Type=RelativeToParent,
              Property=Width, Factor=0.5, Constant=-50}"
              RelativeLayout.YConstraint = "{ConstraintExpression
Type=RelativeToParent,
              Property=Height, Factor=0.5, Constant=-150}"
        />
        <BoxView Color="Blue"
             RelativeLayout.XConstraint = "{ConstraintExpression
Type=RelativeToView, ElementName=lbl,
             Property=X, Factor=1, Constant=-30}"
              RelativeLayout.YConstraint = "{ConstraintExpression
Type=RelativeToView, ElementName=lbl,
              Property=Y, Factor=1, Constant=30}"
 RelativeLayout.WidthConstraint = "150"
 RelativeLayout.HeightConstraint = "100"
        />
</RelativeLayout>
```

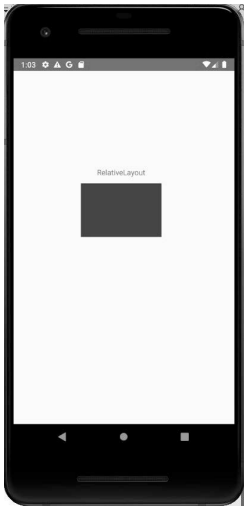


Рис. 4.3. Інтерфейс для прикладу 4.5.

У цьому випадку позиціонування елемента Label встановлюється щодо контейнера RelativeLayout. А елемент BoxView встановлюється щодо елемента

Label. У разі BoxView.X = lbl.X * 1 - 30. Аналогічно обчислюється положення по осі Y.

Для позиціонування елемента в коді C# щодо інших елементів можна використовувати метод Constraint.RelativeToView() замість Constraint.RelativeToParent().

Наприклад, попередній приклад на C#.

Приклад 4.6.

```
public MainPage()
            {
      InitializeComponent();

      RelativeLayout relativeLayout = new RelativeLayout();

      Label label = new Label { Text = "RelativeLayout" };

      BoxView blueBox = new BoxView { BackgroundColor = Color.Blue };

      relativeLayout.Children.Add(label,
        Constraint.RelativeToParent((parent) =>
        {
          return parent.Width * 0.5 - 50;
         //встановлення координати X
        }),
        Constraint.RelativeToParent((parent) =>
        {
          return parent.Height * 0.5 - 150;
         //встановлення координати Y
        })
      );
      relativeLayout.Children.Add(blueBox,
        Constraint.RelativeToView(label, (parent, view1) =>
        {
          return label.X - 30;
          //встановлення координати X
        }),
        Constraint.RelativeToView(label, (parent, view1) =>
        {
          return label.Y + 30;
         //встановлення координати Y
        }),
        Constraint.Constant(150), // встановлення ширини
        Constraint.Constant(100)  // встановлення висоти
      );
      Content = relativeLayout;
          }
```

У Xamarin.Forms BoxView об'єкти, де не вказано розміри, автоматично мають розмір 40x40.

Анонімна функція, яка використовується в цьому методі, приймає два параметри: parent та view1. Параметр parent є батьківським елементом RelativeLayout, а view1 — це елемент, щодо якого потрібно позиціонувати поточний елемент.

У прикладі view1 — це параметр, що представляє елемент label, щодо якого обчислюється позиція для blueBox. Функція використовує view1 для визначення, де має розташовуватися blueBox щодо label, хоча в даному конкретному випадку звернення безпосередньо до label через його змінну, і не використовується переданий параметр view1.

До методу Constraint.RelativeToView() передається елемент, щодо якого задається позиціонування (тобто label), і далі передається делегат, вхідні параметри якого - контейнер та елемент.

Можна так записати делегат в цьому випадку:

```
(parent, view) =>
        {
          return view.X - 30;  //встановлення координати X
              }
```

Контрольні питання

1. Що таке RelativeLayout і для чого він використовується в розробці інтерфейсів?

2. Як відбувається відносне позиціонування елементів всередині RelativeLayout?

3. Які основні відмінності між встановленням розмірів елементів через властивості HeightRequest та WidthRequest порівняно з використанням RelativeLayout.HeightConstraint та RelativeLayout.WidthConstraint?

4. Як можна визначити розмір елемента як відсоток від розміру батьківського контейнера у RelativeLayout?

5. Дайте приклад, як в XAML задати елемент, що має відносне розташування до іншого елемента в межах того ж RelativeLayout.

6. Як в коді C# встановити позицію елемента відносно іншого елемента всередині RelativeLayout з використанням Constraint.RelativeToView()?

Завдання до лабораторної роботи

1. Розробіть інтерфейс для калькулятора за допомогою елемента компонування RelativeLayout на мові XAML. При установці розмірів елементів у RelativeLayout використовувати дві стратегії (встановити значення через властивості HeightRequest та WidthRequest у елементів та встановити

обмеження RelativeLayout.HeightConstraint та RelativeLayout.WidthConstraint). Позиціонування всередині RelativeLayout необхідно встановити щодо контейнера RelativeLayout та щодо інших елементів.

2. Додайте обробники подій для кнопок та реалізуйте логіку калькулятора за допомогою C#.

3. Розробіть інтерфейс для калькулятора за допомогою елемента компонування RelativeLayout на мові C#, також при установці розмірів елементів у RelativeLayout використовувати дві стратегії. Позиціонування всередині RelativeLayout також необхідно встановити щодо контейнера RelativeLayout та щодо інших елементів. Реалізуйте логіку калькулятора на C#.

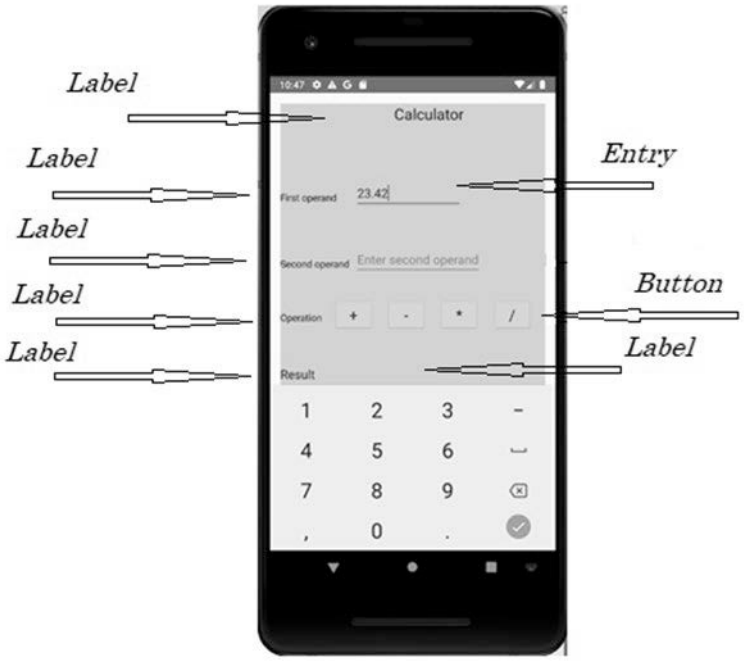


Рис. 4.4. Інтерфейс до лабораторної роботи №4.

ЛАБОРАТОРНА РОБОТА №5
ВИВЧЕННЯ КОНТЕЙНЕРІВ КОМПОНУВАННЯ. ЕЛЕМЕНТ GRID

Мета: ознайомитись з принципами роботи та можливостями використання контейнера Grid в рамках системи компонування інтерфейсу користувача; набути практичних навичок в розміщенні елементів інтерфейсу у сітці за допомогою контейнера Grid.

Теоретичні відомості

Крім звичайних елементів типу кнопок і текстових полів, у Xamarin Forms також є контейнери, які дозволяють скомпонувати вміст, розташувати його певним чином.

Контейнер Grid в Xamarin є потужним інструментом для компонування інтерфейсу користувача. Він дозволяє розміщувати елементи управління у вигляді таблиці з осередками, що спрощує створення складних макетів та гнучке налаштування їх розташування.

Контейнер Grid в Xamarin працює подібно до табличного макету, де елементи розташовуються в осередках, заданих рядками і стовпцями. Це дозволяє створювати макети з різною складністю та налаштовувати розміри, вирівнювання та відступи кожного осередку.

За допомогою властивостей RowDefinitions та ColumnDefinitions створюються набори рядків та стовпців у Grid. Висота рядків та ширина стовпців задається за допомогою GridLength.

Приклад 5.1. Створення Grid на C#:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace AppGrid
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
          Grid grid = new Grid
            {
                RowDefinitions =
            {
                new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) },
```

```
                new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) },
            },
                ColumnDefinitions =
            {
                new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) },
                new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) },
                new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) },
            }
            };
            grid.Children.Add(new BoxView { Color = Color.Red }, 0, 0);
            grid.Children.Add(new BoxView { Color = Color.Blue }, 1, 0);
            grid.Children.Add(new BoxView { Color = Color.Yellow}, 2, 0);
// 2-column, 0 - row

           grid.Children.Add(new BoxView { Color = Color.Teal },0,1);
           grid.Children.Add(new BoxView { Color = Color.Green },1,1);
           grid.Children.Add(new BoxView { Color = Color.Purple },  2,1);

            Content = grid;
        }
    }
}
```

Тут за допомогою RowDefinitions і ColumnDefinitions задаються 2 рядки і три стовпці, тобто у результаті 6 осередків.

Для висоти рядків та ширини стовпців задається значення new GridLength(1, GridUnitType.Star). Число 1 вказує на те, що даний стовпець або рядок буде займати одну частку від усього простору (оскільки всі 2 рядки або 3 стовпці мають значення 1, то весь простір умовно дорівнюватиме 1+1=2 або 1+1+1=3 , а один рядок займатиме 1/2, і стовпець займатиме 1/3). А параметр GridUnitType.Star вказує, що розміри обчислюватимуться пропорційно.

Щоб додати елемент у певну комірку, потрібно вказати номер стовпця та рядки:

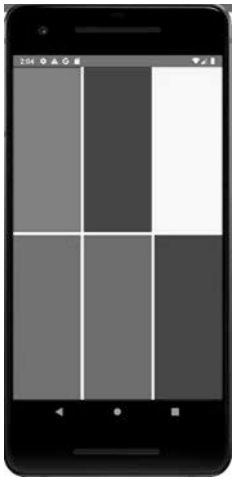
`grid.Children.Add(new BoxView { Color = Color.Blue }, 1, 0);` - елемент BoxView додається до комірки таблиці, яка знаходиться на перетині стовпця 1 і рядка 0 (обчислення починається з 0).

Приклад 5.2. Створення Grid 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="AppGrid.MainPage">
    <Grid>
        <Grid.ColumnDefinitions>
```

```
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <BoxView Color="Red"    Grid.Column="0" Grid.Row="0" />
        <BoxView Color="Teal"   Grid.Column="0" Grid.Row="1" />

        <BoxView Color="Blue"   Grid.Column="1" Grid.Row="0" />
        <BoxView Color="Green"  Grid.Column="1" Grid.Row="1" />

        <BoxView Color="Yellow" Grid.Column="2" Grid.Row="0" />
        <BoxView Color="Purple" Grid.Column="2" Grid.Row="1" />

    </Grid>
</ContentPage>
```

Використання зірочки в XAML аналогічно до параметра



GridUnitType.Star.

Рис. 5.1. Інтерфейс для прикладів 5.1 та 5.2.

Крім пропорційних розмірів можна задати автоматичні або абсолютні розміри.

Приклад 5.3. Задання автоматичних та абсолютних розмірів у Grid 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="AppGrid.MainPage">

    <Grid ColumnSpacing="5">
        <Grid.RowDefinitions>
```

```
            <RowDefinition Height="*" />
            <RowDefinition Height="200" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="50" />
        </Grid.ColumnDefinitions>

        <BoxView Color="Red" Grid.Column="0" Grid.Row="0" />
        <BoxView Color=" Teal " Grid.Column="0" Grid.Row="1" />

        <BoxView Color=" Blue " Grid.Column="1" Grid.Row="0" />
        <BoxView Color="Green" Grid.Column="1" Grid.Row="1" />

        <BoxView Color=" Yellow " Grid.Column="2" Grid.Row="0" />
        <BoxView Color=" Purple " Grid.Column="2" Grid.Row="1" />

    </Grid>
</ContentPage>
```

У разі визначення таблиці в Xamarin.Forms з використанням ColumnDefinition та встановленням ширини колонки рівної Auto, ширина колонки буде автоматично визначена на основі її вмісту.

У разі визначення таблиці в Xamarin.Forms з використанням ColumnDefinition та встановленням ширини колонки, яка дорівнює 50, ширина колонки буде фіксованою і складатиме 50 одиниць вимірювання. Одиниця вимірювання залежить від пристрою та налаштувань екрана, але за замовчуванням це пікселі (px).

У разі визначення таблиці в Xamarin.Forms з використанням ColumnDefinition та встановленням ширини колонки, рівної "*", ширина колонки буде автоматично визначена на основі доступного простору, що залишився.

Це означає, що якщо в таблиці визначено колонки з шириною "*", то вони будуть рівномірно розподілені за доступним простором, пропорційно їх ширині. Наприклад, якщо в таблиці визначено дві колонки із шириною "*", то кожна з них займатиме 50% доступного простору.

Приклад 5.4. Визначення різних типів розмірів у Grid у коді C#:

```
public MainPage()
{
    InitializeComponent();
    Grid grid = new Grid
    {
        RowDefinitions =
```

```
                {
                    new RowDefinition { Height = new GridLength(2,
GridUnitType.Star) },
                    new RowDefinition { Height = 200 }
                },
                ColumnDefinitions =
                {
                    new ColumnDefinition { Width = GridLength.Auto },
                    new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) },
                    new ColumnDefinition { Width = 50 }
                }
            };
            grid.Children.Add(new BoxView { Color = Color.Red }, 0, 0);
            grid.Children.Add(new BoxView { Color = Color.Blue }, 1, 0);
            grid.Children.Add(new BoxView { Color = Color.Yellow },2, 0);
            // 2-column, 0 - row
            grid.Children.Add(new BoxView { Color = Color.Teal }, 0, 1);
            grid.Children.Add(new BoxView { Color = Color.Green }, 1, 1);
            grid.Children.Add(new BoxView { Color = Color.Purple },2, 1);
            Content = grid;
}
```

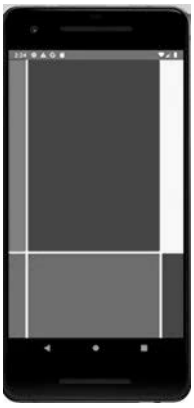


Рис. 5.2. Інтерфейс для прикладів 5.3 та 5.4.

Клас Grid визначає дві спеціальні властивості для створення відступів:

– ColumnSpacing: визначає простір між стовпцями;

– RowSpacing: визначає простір між рядками.

Приклад 5.5. Створення відступів у Grid 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="AppProbaGrid.MainPage">

    <Grid ColumnSpacing="5">
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
```

```
            <RowDefinition Height="200" />
        </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="50" />
    </Grid.ColumnDefinitions>

    <BoxView Color="Red" Grid.Column="0" Grid.Row="0" />
    <BoxView Color=" Teal " Grid.Column="0" Grid.Row="1" />

    <BoxView Color=" Blue " Grid.Column="1" Grid.Row="0" />
    <BoxView Color="Green" Grid.Column="1" Grid.Row="1" />

    <BoxView Color=" Yellow " Grid.Column="2" Grid.Row="0" />
    <BoxView Color=" Purple " Grid.Column="2" Grid.Row="1" />
</Grid>
</ContentPage>
```

Приклад 5.6. Створення відступів у Grid на C#:

```
public MainPage()
        {
            InitializeComponent();
            Grid grid = new Grid
            {
                ColumnSpacing = 5,
                RowDefinitions =
                {
                    new RowDefinition { Height = new GridLength(2,
GridUnitType.Star) },
                    new RowDefinition { Height = 200 }
                },
                ColumnDefinitions =
                {
                    new ColumnDefinition { Width = GridLength.Auto },
                    new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) },
                    new ColumnDefinition { Width = 50 }
                }
            };

            grid.Children.Add(new BoxView { Color = Color.Red }, 0, 0);
            grid.Children.Add(new BoxView { Color = Color.Blue }, 1, 0);
            grid.Children.Add(new BoxView { Color = Color.Yellow },2, 0);
            // 2-column, 0 - row

            grid.Children.Add(new BoxView { Color = Color.Teal }, 0, 1);
            grid.Children.Add(new BoxView { Color = Color.Green }, 1, 1);
            grid.Children.Add(new BoxView { Color = Color.Purple },2, 1);
```

```
            Content = grid;

        }
```

Об'єднання осередків: за допомогою властивості ColumnSpan можна поєднати кілька стовпців, а за допомогою властивості RowSpan - поєднати осередки.

Приклад 5.7. Об'єднання осередків у Grid на C#:

```
public MainPage()
        {
            InitializeComponent();

            Grid grid = new Grid
            {
                ColumnSpacing = 5 ,
                RowDefinitions =
                {
                    new RowDefinition { Height = new GridLength(2,
GridUnitType.Star) },
                    new RowDefinition { Height = 200 }
                },
                ColumnDefinitions =
                {
                    new ColumnDefinition { Width = GridLength.Auto },
                    new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) },
                    new ColumnDefinition { Width = 50 }
                }
            };

        BoxView redBox = new BoxView { Color = Color.Red };
        BoxView blueBox = new BoxView { Color = Color.Blue };
        BoxView yellowBox = new BoxView { Color = Color.Yellow };
        BoxView TealBox = new BoxView { Color = Color.Teal };
        BoxView PurpleBox = new BoxView { Color = Color.Purple };
        grid.Children.Add(redBox, 0, 0);
        grid.Children.Add(blueBox, 1, 0);
        grid.Children.Add(yellowBox, 2, 0);
        grid.Children.Add(TealBox, 0, 1);
        grid.Children.Add(PurpleBox, 1, 1);

        Grid.SetColumnSpan(PurpleBox, 2);   //  тягнемо на два стовпці
        Content = grid;
    }
```

Рис. 5.3. Інтерфейс для прикладів 5.5 та 5.6.

Приклад 5.8. Об'єднання осередків у Grid засобами XAML:

```
<Grid ColumnSpacing="5">
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="200" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="50" />
        </Grid.ColumnDefinitions>

        <BoxView Color="Red" Grid.Column="0" Grid.Row="0" />
        <BoxView Color="Purple" Grid.Column="1" Grid.Row="1"
Grid.ColumnSpan="2" />

        <BoxView Color="Blue" Grid.Column="1" Grid.Row="0" />
        <BoxView Color="Teal" Grid.Column="0" Grid.Row="1" />
        <BoxView Color="Yellow" Grid.Column="2" Grid.Row="0" />
 </Grid>
```

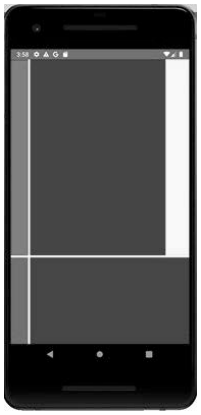


Рис. 5.4. Інтерфейс для прикладів 5.7 та 5.8.

Контрольні питання

1. Опишіть призначення та основні властивості контейнера Grid.
2. Як визначити кількість рядків та стовпців у контейнері Grid?
3. Для чого використовуються властивості RowSpan та ColumnSpan у контексті елемента Grid?

Завдання до лабораторної роботи

Завдання 1

1. Розробіть інтерфейс для калькулятора (рис. 5.5) за допомогою елемента компонування Grid на мові XAML.

2. Додайте обробники подій для кнопок та реалізуйте логіку калькулятора за допомогою C#.

3. Розробіть інтерфейс для калькулятора за допомогою елемента компонування Grid на мові C# і реалізуйте логіку калькулятора на C#.

Завдання 2

1. Розробіть інтерфейс за допомогою елемента компонування Grid на мові XAML відповідно варіанту. Вимога – дотримуватись пропорцій осередків. Встановити як фон для кожного осередку свій колір. (Номер варіанту співпадає з номером студента в списку групи).

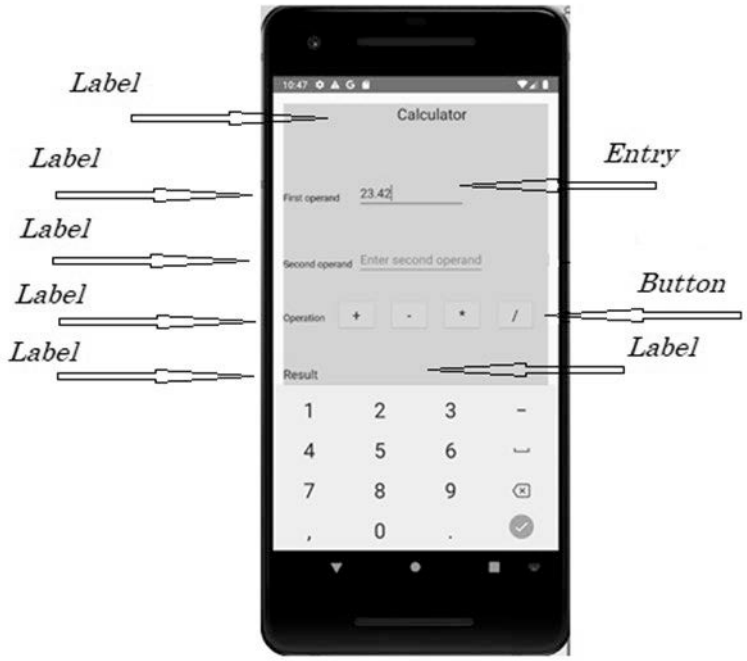
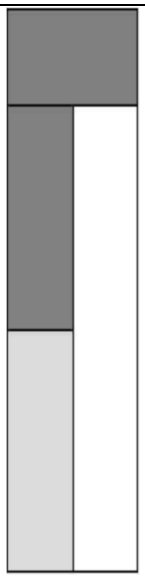
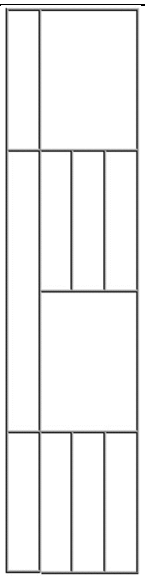
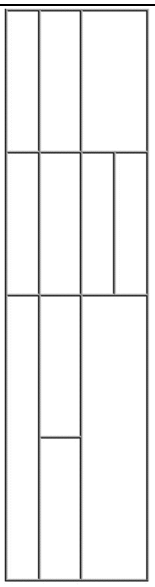
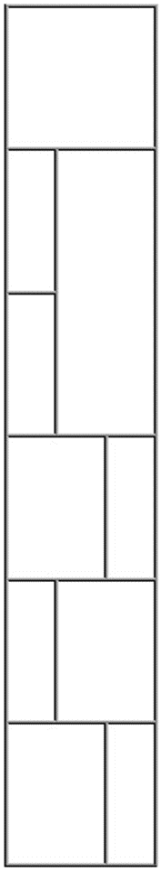
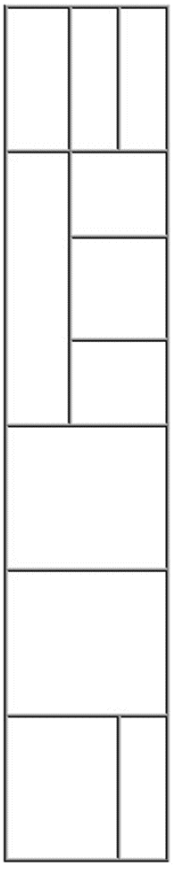
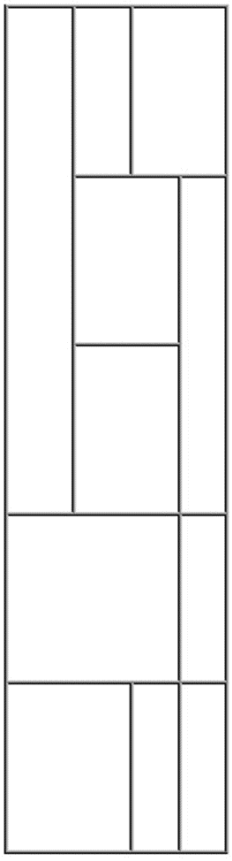
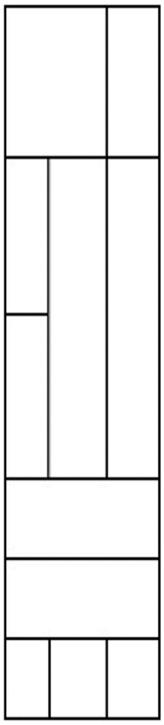
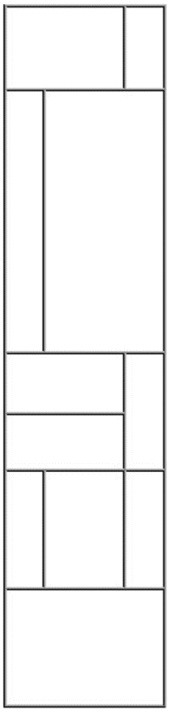
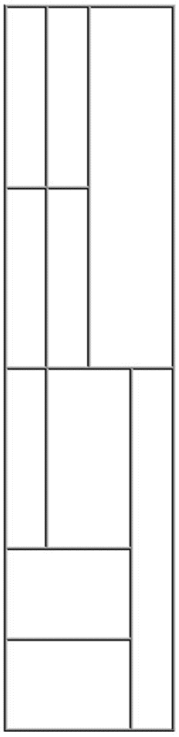
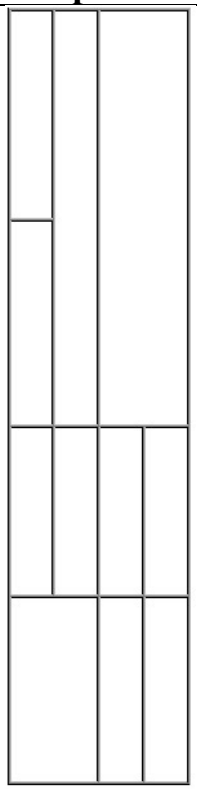
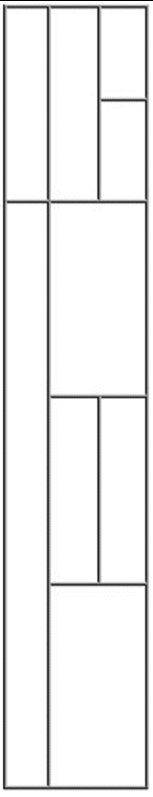
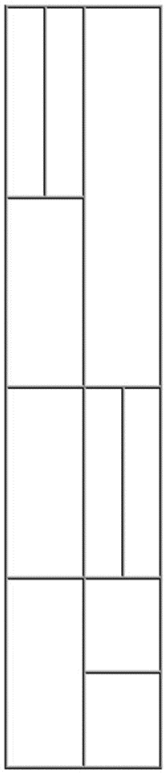


Рис. 5.5. Інтерфейс для калькулятора

2. Розробіть інтерфейс за допомогою елемента компонування Grid на мові C#.

Варіант 1	Варіант 2	Варіант 3
		
Варіант 4	**Варіант 5**	**Варіант 6**
		

Варіант 7	Варіант 8	Варіант 9
		
Варіант 10	**Варіант 11**	**Варіант 12**
		

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 контроль потребує вимірювання, щоб повідомити розмір відповідно до його вмісту та своїх властивостей.

OnSizeAllocated(Double, Double) - викликається, коли контроль 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.

Специфічні контейнери і лейаути:

6. **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.

7. **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.

8. **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.

9. **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 в 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.

10. **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.

Основи сторінок та навігації:

11. **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 1

Методичні вказівки до лабораторних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальностей 122 «Комп'ютерні науки»,
123 «Комп'ютерна інженерія»,
151 «Автоматизація та комп'ютерно-інтегровані технології»

Укладачі

Камєнєва Алла Вікторівна

Зуй Оксана Миколаївна

Зудіхін Єгор Олексійович

В авторській редакції

Підписано до друку 01.06.2024 р.
Формат 60x84/16. Папір офсетний.
Цифровий друк. Гарнітура Times.
Ум. друк. арк. 3,95. Наклад 30 пр.
Замовлення № 1024-08.

Видавництво: Олді+
65101, м. Одеса, вул. Інглезі, 6/1,
тел.: +38 (095) 559-45-45,
e-mail: office@oldiplus.ua
Свідоцтво ДК № 7642 від 29.07.2022 р.
Замовлення книг:
тел.: +38 (050) 915-34-54, +38 (068) 517-50-33
e-mail: book@oldiplus.ua