Міністерство освіти і науки України
Харківський національний автомобільно-дорожній університет

Механічний факультет

Кафедра комп’ютерних наук і інформаційних систем

Методичні вказівки до практичних робіт з дисципліни «Кросплатформне програмування» для бакалаврів спеціальності F3 Комп’ютерні науки галузі знань F Інформаційні технології

Харків – 2025

УДК 004.43

Лебединський А. В. Методичні вказівки до практичних робіт з дисципліни «Кросплатформне програмування» для бакалаврів спеціальності «F3 Комп’ютерні науки». Харків: ХНАДУ, 2025. 107 с.

У методичних вказівках наведено основні підходи до організації та проведення практичних занять з дисципліни «Кросплатформне програмування». Матеріал спрямований на формування у студентів фахових компетентностей у сфері сучасних технологій розробки програмного забезпечення, що функціонує на різних апаратних платформах та операційних системах. Методичні вказівки призначені для здобувачів вищої освіти освітнього рівня «Бакалавр» спеціальності F3 «Комп’ютерні науки».

ПРАКТИЧНА РОБОТА №1

Тема роботи: дослідження можливостей інтегрованого середовища розробки Visual Studio для створення простих кросплатформних додатків на .NET MAUI.

Мета роботи: дослідити можливості інтегрованого середовища розробки Visual Studio і отримати практичні навички створення простих кросплатформних додатків з використанням .NET Multi-platform App UI (MAUI).

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Сучасна мобільна екосистема характеризується значною фрагментацією, де домінують дві провідні операційні системи – Android та iOS. Статистичні дослідження демонструють, що переважна більшість мобільних застосунків розробляється для декількох платформ одночасно, що відображає прагнення компаній максимізувати охоплення цільової аудиторії через різноманітні пристрої та операційні системи.

Традиційний підхід до розробки мобільних застосунків супроводжується низкою фундаментальних викликів. Архітектурні відмінності у побудові користувацьких інтерфейсів вимагають від розробників адаптації застосунків під специфічні вимоги кожної платформи, що значно ускладнює процес розробки та підтримки коду. Різноманітність програмних інтерфейсів та їх специфічних реалізацій потребує глибокого розуміння унікальних особливостей API кожної платформи, що підвищує поріг входження для розробників.

Технологічна гетерогенність розробницьких середовищ створює додаткові бар'єри для ефективної розробки. Створення застосунків для iOS традиційно вимагає використання macOS разом із спеціалізованими інструментами, такими як Xcode, та застосування мов програмування Swift або Objective-C. Розробка для Android передбачає використання різноманітних середовищ розробки, включаючи Android Studio, із переважним застосуванням

мови Java або Kotlin. Ця технологічна роздробленість призводить до необхідності підтримки різних команд розробників із специфічними навичками для кожної платформи.

1.1 Еволюція до .NET MAUI

.NET Multi-platform App UI (MAUI) представляє собою еволюційний крок у кросплатформній розробці мобільних застосунків, що виник як наступник технології Xamarin.Forms. Ця сучасна платформа дозволяє розробникам створювати єдину кодову базу з використанням C# та екосистеми .NET, яка функціонує на множині платформ, включаючи Android, iOS, Windows та macOS.

Архітектурна концепція .NET MAUI ґрунтується на принципі «write once, run anywhere», що кардинально змінює парадигму мобільної розробки. Платформа забезпечує високий рівень абстракції над нативними API операційних систем, одночасно зберігаючи можливість прямого доступу до платформо-специфічних функціональностей коли це необхідно.

1.2 Архітектурні переваги .NET MAUI

Впровадження .NET MAUI в процес розробки мобільних застосунків надає розробникам можливість створення уніфікованої кодової бази, що значно спрощує процеси розробки, тестування та підтримки програмного продукту. Ця єдина кодова база дозволяє розробникам зосередитися на бізнес-логіці застосунку, а не на технічних особливостях різних платформ.

Платформа забезпечує безпосередній доступ до нативних API кожної цільової операційної системи, що гарантує високу продуктивність та можливість використання всіх специфічних можливостей пристроїв. Такий підхід дозволяє створювати застосунки, які не поступаються за функціональністю та продуктивністю нативним рішенням.

Використання мови програмування C# та екосистеми .NET забезпечує розробникам доступ до потужних інструментів розробки, багатої бібліотеки класів та розвиненої спільноти розробників. Це особливо важливо в контексті сучасних тенденцій, де швидкість розробки та якість кінцевого продукту є критичними факторами успіху.

1.3 Компонентна архітектура платформи

.NET MAUI складається з декількох ключових компонентів, кожен з яких відповідає за специфічні аспекти кросплатформної розробки. Основою платформи є єдиний фреймворк, який забезпечує абстракцію над різними операційними системами та їх специфічними особливостями.

Платформа включає спеціалізовані бібліотеки для кожної цільової операційної системи. Компонент для Android забезпечує інтеграцію з Android SDK та доступ до специфічних можливостей платформи Google. Модуль для iOS надає повну інтеграцію з iOS SDK та можливості екосистеми Apple. Підтримка Windows дозволяє створювати сучасні десктопні застосунки з використанням WinUI 3, а компонент для macOS забезпечує нативну інтеграцію з операційною системою Apple для настільних комп'ютерів.

Ключовою особливістю .NET MAUI є можливість створення як платформо-специфічних застосунків, так і повністю кросплатформних рішень з єдиною кодовою базою. Така гнучкість дозволяє розробникам обирати оптимальну стратегію розробки залежно від специфічних вимог проекту та цільової аудиторії.

Сучасні тенденції в розробці мобільних застосунків демонструють зростаючу популярність кросплатформних рішень, таких як .NET MAUI, що обумовлено потребою в оптимізації ресурсів розробки та скороченні часу виходу продукту на ринок. Ця технологія представляє собою відповідь на виклики сучасного мобільного розвитку, пропонуючи елегантне та ефективне рішення для створення високоякісних мобільних застосунків.

1.4 Різниця між «Проєктом» (Project) та «рішенням» (Solution) у Visual Studio

У Visual Studio проєкт – це одиниця, яку можна зібрати в конкретний артефакт: виконуваний файл (.exe), бібліотеку (.dll) або вебзастосунок. Він містить вихідний код, ресурси (наприклад, зображення), налаштування збірки і залежності (NuGet-пакети). Технічно все це описано у файлі проєкту (наприклад, .csproj), який є декларативною інструкцією для MSBuild: що і як компілювати, які цільові фреймворки використовувати, які властивості застосувати у Debug/Release, тощо. Проєкт можна збирати автономно – це зручно для CI/CD, модульних тестів і розділення відповідальностей між командами.

Рішення – це контейнер для одного або кількох проєктів. Воно визначає, які проєкти входять до спільного робочого простору, як вони організовані в Solution Explorer, і допомагає керувати спільною побудовою та налагодженням. Структура рішення зберігається у файлі .sln, а індивідуальні робочі параметри розробника (наприклад, розташування вікон, точки зупину) – у прихованому .suo. Рішення може містити «папки рішення» для логічного впорядкування без зміни фізичних шляхів у файловій системі.

Практично: проєкт відповідає за технічні деталі конкретного артефакту (його код, залежності, конфігурації), а рішення – за організацію всієї «екосистеми» навчального або комерційного продукту (застосунок, бібліотеки спільного коду, тести, інструменти міграцій). Коли створюється новий проєкт у чистому середовищі, Visual Studio автоматично створює для нього рішення. У реальних курсах і командах зручно мати одне рішення з кількома проєктами: наприклад, основний застосунок, проєкт юніт-тестів і окрему бібліотеку з бізнес-логікою.

Що варто запам’ятати: працюючи з помилками збірки або налаштуваннями таргетів (TargetFramework, константи компілятора), шукайте

їх у файлі проєкту – це джерело правди для збірки. Якщо потрібно організувати навчальний репозиторій, підключити/відключити кілька модулів або запускати спільне налагодження – це рівень рішення. Рішення упорядковує і пов’язує, проєкт – збирає і постачає.

2 ПІДГОТОВЧА ЧАСТИНА

2.1 Встановлення Visual Studio 2022

Щоб розпочати розробку на .NET MAUI у Visual Studio 2022, спершу потрібно встановити Visual Studio 2022 Community за посиланням https://visualstudio.microsoft.com/downloads/. На офіційному сайті потрібно вибрати версію «Community», яка розрахована на безкоштовне користування. Функціонал безкоштовної версії цілком достатньо не тільки для знайомства та написання програм на .MAUI (і не тільки), а й для написання повноцінних потужних програмних продуктів майже у будь-якій сфері IT. У процесі встановлення погоджуємося із усіма налаштуваннями інсталятора.

Після встановлення, програмний пакет Visual Studio 2022 буде поділений на дві програми: Visual Studio 2022 Installer та сам Visual Studio 2022. Перший (рис. 2.1) потрібен для встановлення додаткового робочого навантаження, а саме компонентів під .NET MAUI.

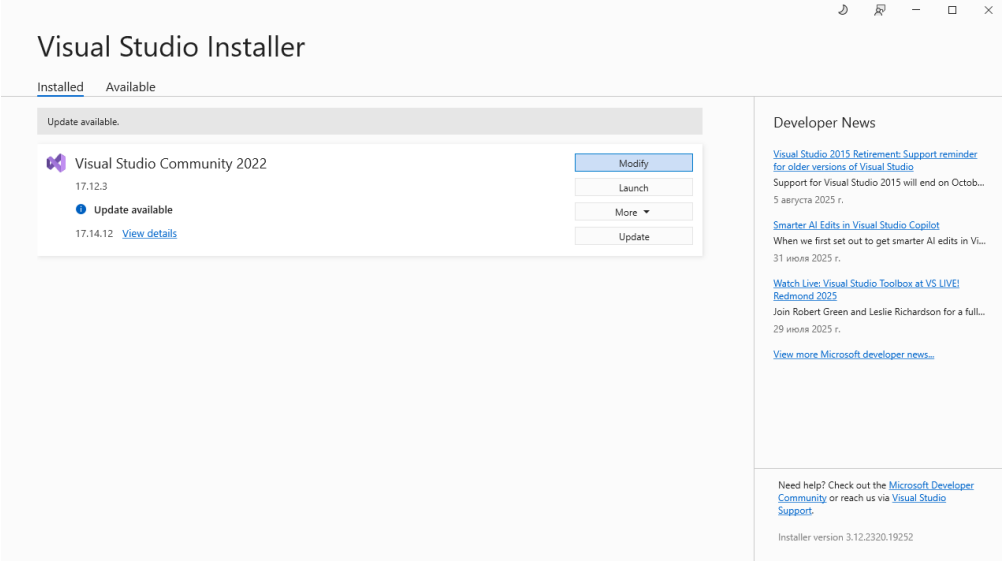


Рисунок 2.1 – Зовнішній вигляд вікна Visual Studio Installer

Друга програма є основною для створення, зміни та видалення проєктів, які написані у програмному середовищі розробки Visual Studio 2022.

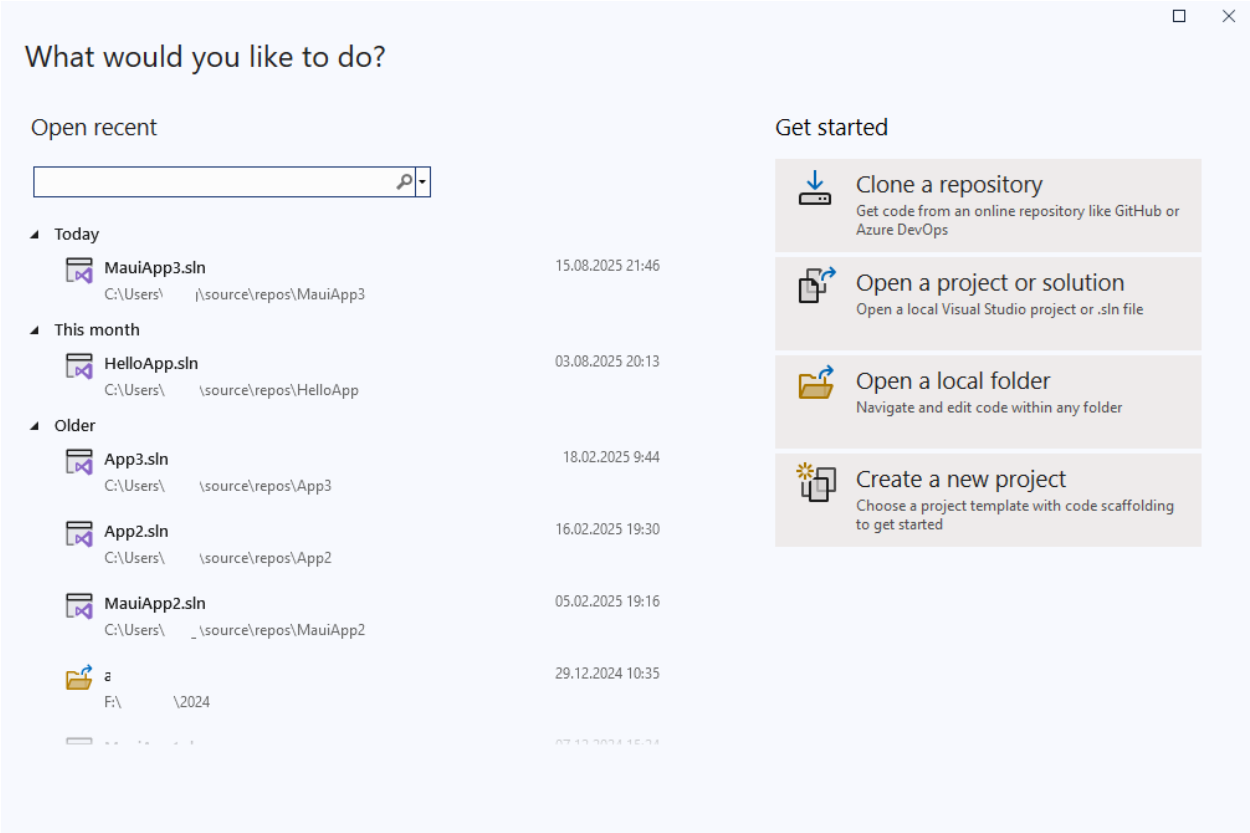


Рисунок 2.2 – Зовнішній вигляд початкового вікна Visual Studio 2022

На рисунку 2.1 після натискання кнопки «Modify» («Модифікувати») заходимо у меню для встановлення робочого навантаження .NET MAUI для нашого середовища розробки Visual Studio 2022. На рисунку 2.3 показано стрілочкою, що потрібно встановити, а саме: у категорії «Desktop & Mobile» ставимо галочку на пункт **.NET Multi-platform App UI development**. Як можна бачити, у описі пункту написано, що ця технологія дозволяє будувати кросплатформні додатки на Android, iOS, Windows and Mac, використовуючи мову програмування C#. Це автоматично додає SDK і засоби для розробки під Android, iOS/Mac Catalyst та Windows, а також елементи емуляції Android. Для повноцінного досвіду на Windows корисно також мати активним робоче навантаження «.NET Desktop Development» для діагностики й суміжних інструментів. Після підтвердження вибору («Install while downloading») запускається встановлення; за потреби інсталятор завантажить додаткові залежності (Android SDK, OpenJDK тощо) і може запропонувати перезавантаження системи.

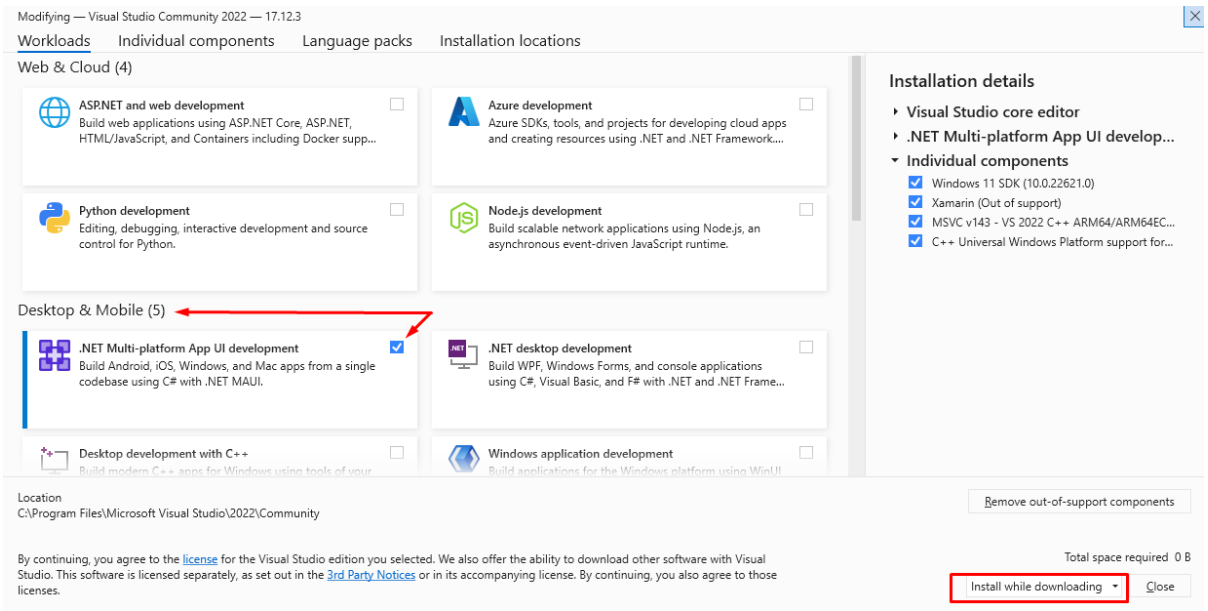


Рисунок 2.3 – Встановлення потрібного робочого навантаження

Після завершення встановлення Visual Studio відкривається перший раз, виконується вхід в обліковий запис (за наявності підписки/синхронізації

налаштувань), а також перевіряється доступність шаблонів «.NET MAUI App» у вікні створення нового проекту. Якщо шаблонів немає, через Visual Studio Installer можна додатково увімкнути робоче навантаження .NET MAUI або оновити інсталяцію до актуальної версії. За потреби в подальшому окремі платформи (Android/iOS/Mac/Windows) можна додавати через модифікацію інсталяції.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Створення та налаштування проєкту на .NET MAUI

Для створення першого проєкту необхідно запустити встановлений Visual Studio 2022 з меню «Пуск». Після запуску програми у полі пошуку технологій розробки вписуємо «maui» (рис. 3.1) і вибираємо перший шаблон у списку, а саме: «.NET MAUI App». У описі можна побачити, що цей шаблон дозволяє створити .NET MAUI кросплатформний застосунок. Далі у вікні «Configure your new project» вказати назву проєкту та теку розміщення рішення, після чого натиснути «Next». Ім’я можна надати у звичному форматі, наприклад MauiApp, як у офіційному туторіалі. У вікні «Additional information» обрати цільову версію .NET (наприклад, .NET 9.0, якщо доступно у встановленій конфігурації) і натиснути «Create» («Створити»). Visual Studio створить рішення та ініціює відновлення залежностей NuGet; дочекатися повідомлення про завершення (Restored/Ready) в статус-барі.

3.2 Інтерфейс Visual Studio 2022

3.2.1 Головне вікно та макет робочого середовища

Visual Studio 2022 – 64-бітова IDE з гнучким докінгом панелей, підтримкою багато-вкладкового інтерфейсу й збереженням макетів робочого середовища. Робочий простір складається з центральної області редактора

коду/дизайнера, панелей інструментів по краях (Solution Explorer, Error List, Output тощо), верхнього меню та командних панелей, а також нижнього статусного рядка. Усі вікна можна перетягувати, пришвартовувати до країв, вкладати одне в одне або виносити на другий монітор; конфігурації макетів зберігаються та перемикаються через Window > Manage Window Layouts. Тема, шрифт і контраст синхронізуються з Windows, є вбудовані теми (Light/Dark/Blue) і масштабування UI, що важливо для тривалих сесій.

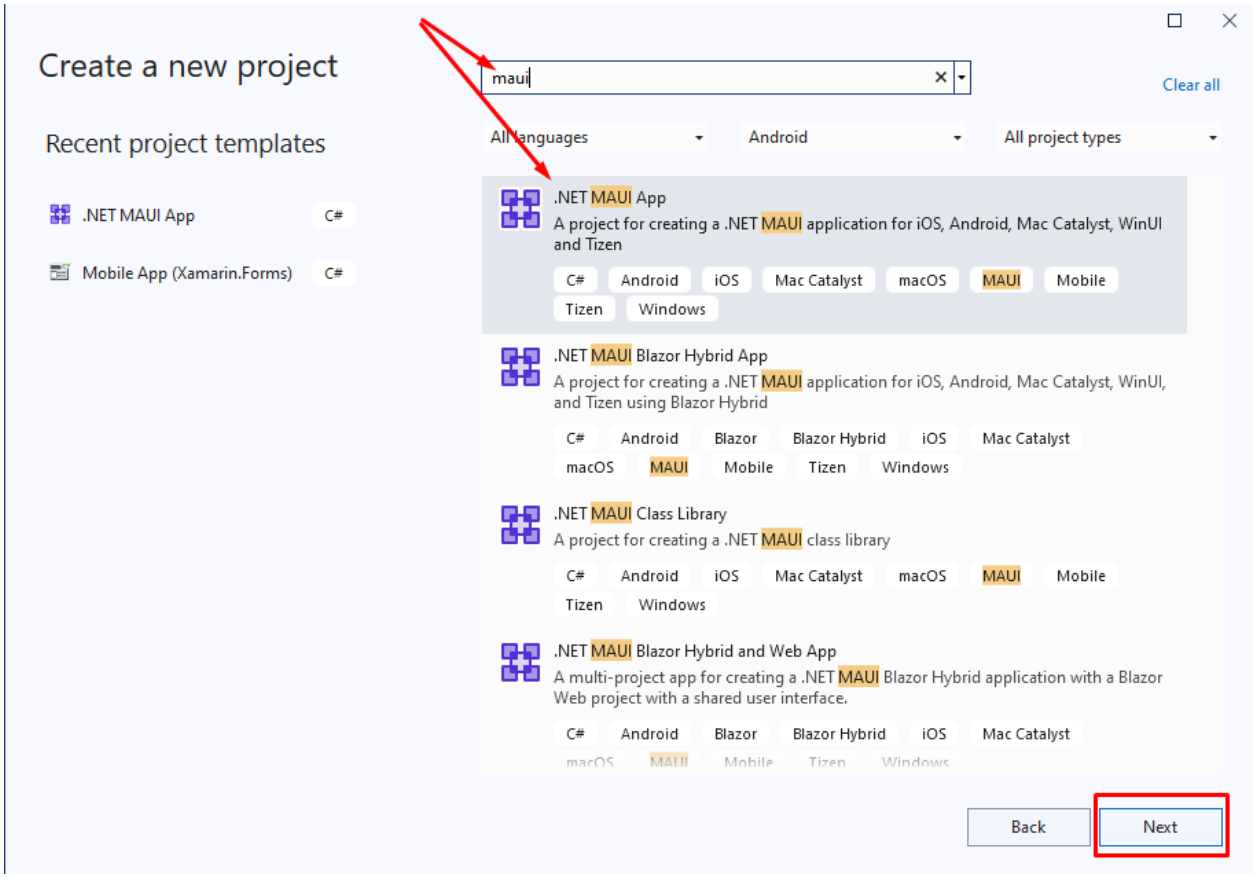


Рисунок 3.1 – Створення нового проєкту .NET MAUI

3.2.2 Головне меню, командні панелі й пошук команд

Верхнє меню (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help) (рис. 3.2) організовує всі дії IDE. Більшість пунктів має контекстні підменю, залежні від типу відкритого файлу/проекту. Панель швидкого доступу містить кнопки для запуску/зупинки, вибору конфігурації (Debug/Release), цільової платформи (x64/Any CPU), стартового

проєкту та профілю запуску. Вбудований пошук (Ctrl+Q) знаходить команди меню, опції налаштувань, файли, типи й символи, а також елементи з маркетплейсу розширень; окремо працюють Go to (Ctrl+T) для навігації по кодовій базі та Find in Files (Ctrl+Shift+F) з фільтрами масок, регістру та цілого слова.

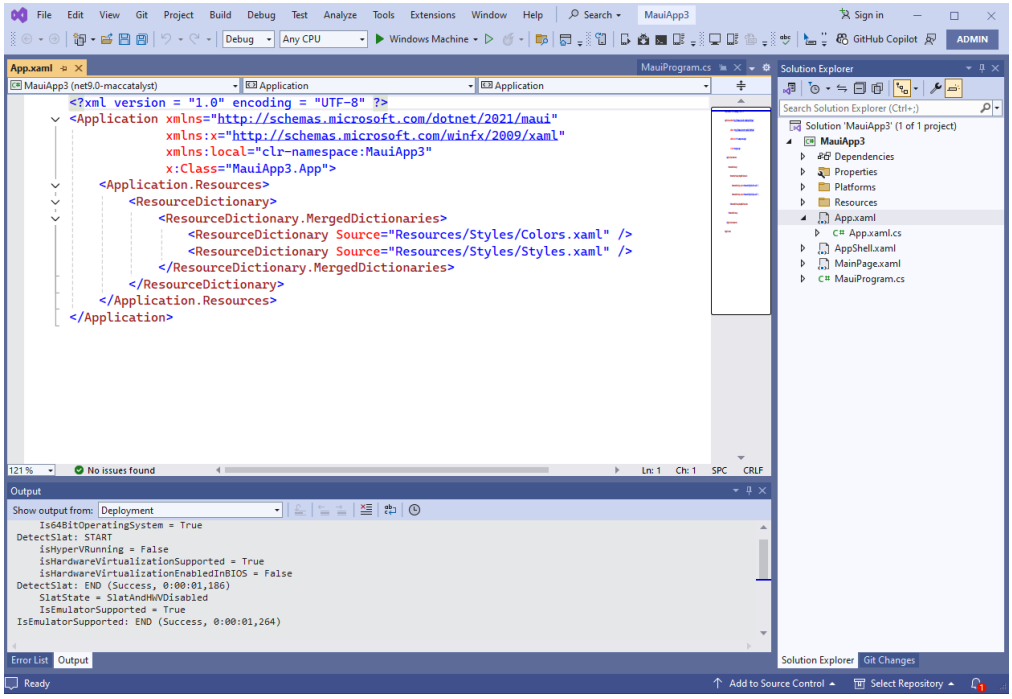


Рисунок 3.2 – Зовнішній вигляд інтерфейсу нового проєкту .NET MAUI

3.2.3 Редактор коду: вкладки, навігація, підказки

Центральний редактор підтримує багато вкладок із попереднім переглядом, групуванням у колонки/ряди та механікою «sticky» вкладок. Підсвітка синтаксису, IntelliSense з параметр-хінтами, автодоповненням і контекстними підказками інтегрується з мовними службами (Roslyn для .NET). Доступні «Peek Definition», «Go to Definition/Implementation», «CodeLens» над оголошеннями (посилання, тести, історія Git), а також Quick Actions (Ctrl+.) для рефакторингів: перейменування, виділення методу, впровадження інтерфейсів, створення типів, упорядкування using, фікси аналізаторів. Для великих файлів працюють структурувальні «region»,

breadcrumb-навігація по просторах імен/типах/членах, і міні-мапа справа для швидкого прокручування.

3.2.4 Панель «Solution Explorer»: рішення, проекти та файли

Solution Explorer відображає ієрархію «Solution→Projects→Folders/Files», з вузлами для залежностей (рис. 3.2), пакетів NuGet, налаштувань та ресурсів. Угорі є інтегрована стрічка пошуку по вузлах і панель дій (показ прихованих файлів, сортування, розгортання, «Home» до кореня). Контекстні меню вузлів дозволяють додавати нові елементи, керувати залежностями, відкривати файл у редакторі/дизайнері, змінювати властивості проекту, налаштовувати стартовий проєкт, підключати сервіси. Для C#/C++ доступні фільтри перегляду (наприклад, лише відкриті файли) та відображення логічної/фізичної структури.

3.2.5. Інтеграція з Git: зміни, гілки, історія

Вбудований Git-досвід охоплює ініціалізацію репозиторію, клонування, підключення до GitHub/Azure DevOps, перегляд Changes та Staged Changes, коміти, pull/push/fetch, управління гілками з графом комітів, конфлікт-мердж-редактор, інлайн-дифи й режим «summary» для концентрації на суттєвих змінах. Є інтегрований Pull Request досвід: створення, перегляд, коментарі прямо з IDE. Статуси гілки, відрив від HEAD і підказки коміт-меседжів відображаються у статусному рядку та панелях Git.

3.2.6 Побудова, помилки та вивід: Build, Error List, Output

Команди Build/ Rebuild/ Clean працюють на рівні рішення/проєкту з окремими конфігураціями та платформами. Вікно Error List агрегує Errors, Warnings, Messages з компілятора й аналізаторів, підтримує фільтри за проєктом/документом/помилкою, сортування та навігацію до відповідних рядків коду. Вікно Output показує журнали збірки, діагностику інструментів, повідомлення пакетів NuGet, Git та ін.

3.2.7 Налагодження: керування, вікна стану та діагностика

Зелена кнопка запуску стартує «Debug» або без нього (Ctrl+F5); конфігуруються ціль, аргументи, змінні середовища, робоча папка, профіль, інструменти діагностики. Під час налагодження доступні «Call Stack, Locals, Autos, Watch, QuickWatch, Immediate, Exception Settings, Threads/Tasks, Parallel Stacks, Disassembly, Memory, Registers. Breakpoints» підтримують умови, фільтри потоків/процесів, дії (лог без зупинки). Є «Edit» and «Continue» для .NET/C++, а також «Hot Reload» для застосунків на .NET і XAML, що дозволяє застосовувати зміни без перезапуску. Performance Profiler, CPU Usage, .NET Object Allocation, Event Viewer, Database/HTTP Tooling допомагають локалізувати вузькі місця.

3.2.8 Редактори UI та дизайнери: XAML, WinForms, Web

Для WPF/UWP/WinUI застосовується XAML-редактор зі спліт-переглядом (XAML/Designer), панеллю Toolbox, вікном Properties і ієрархією елементів, синхронізованих із кодом; XAML Hot Reload прискорює ітерації без повного перезапуску. Для WinForms є візуальний дизайнер форм і редактор властивостей компонентів. Веброзробка використовує редактори Razor/Blazor з підсвіткою, IntelliSense, Live Preview для CSS/JS і інтеграцію з локальними профілями запуску (IIS Express, Kestrel, браузерні таргети).

3.3 Запуск першої MAUI-програми на Windows Machine у новому проєкті

Після створення проєкту .NET MAUI у Visual Studio 2022 потрібно переконатися, що середовище готове до запуску саме на Windows: для цього за замовчуванням достатньо Windows Machine як цілі налагодження, а також встановленого робочого навантаження «.NET Multi-platform App UI» в інсталяторі Visual Studio; у стартовому туторіалі Microsoft крок запуску для

Windows полягає у натисканні кнопки Windows Machine на панелі запуску, що ініціює побудову та виконання застосунку на локальній машині. Якщо це перший запуск на цьому ПК, Visual Studio збере та інсталює необхідні пакети, відновить залежності і побудує стандартну заготовку додатку MAUI (кнопка «Click me» з лічильником) (рис. 3.3), після чого вікно застосунку з'явиться у середовищі Windows і реагуватиме на натискання, збільшуючи лічильник – це підтверджує коректний запуск конвеєра побудови та розгортання для цілі Windows.

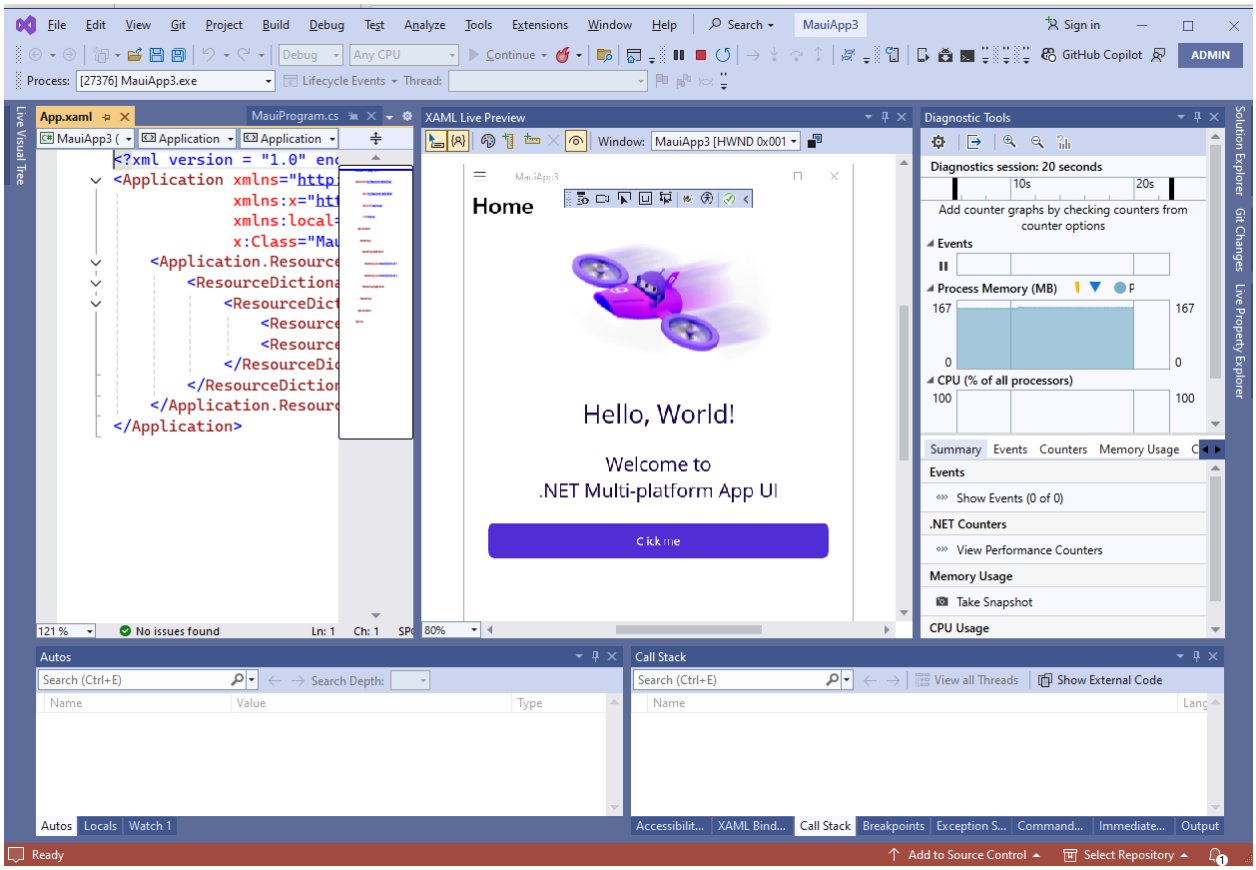


Рисунок 3.3 – Зовнішній вигляд інтерфейсу нового проєкту .NET MAUI під час розгортання на локальній машині «Windows Machine»

Стандартний робочий цикл такий: обирається конфігурація «Debug» та ціль «Windows Machine», після чого натискається F5 або кнопка «Start Debugging», Visual Studio відновлює NuGet-пакети, компілює таргет «netX.Y-windows», запускає процес застосунку і під'єднує налагоджувач; у вікні

«Output/Error List» можна спостерігати повідомлення збірки, а у статусному рядку – поточний профіль і стан розгортання. Якщо запуск не відбувається через невідповідність профілю або фреймворка, потрібно перевірити правильність «Debug Target» (має бути саме «netX-windows») і активний профіль запуску «Windows Machine»; у випадку коли Visual Studio нав'язує профіль, потрібно зняти «Create a Windows MSIX package» у властивостях або вручну виставити «commandName» у launchSettings.json на «Project» для профілю «Windows Machine», щоб повернутись до непакованого сценарію, який не потребує «Developer Mode». Коли застосунок стартує, взаємодія з типовою кнопкою «Click me» демонструє працездатність шаблону MAUI на Windows і підтверджує, що стек WinUI+MAUI налаштований коректно на локальній машині.

3.4 Запуск першої MAUI-програми у Android Simulator у новому проєкті

Після створення проєкту .NET MAUI у Visual Studio 2022 потрібно переконатися, що встановлені Android-інструменти: Android SDK, Android Emulator, потрібні образи системи (наприклад, Android 13), а також включена апаратна віртуалізація у BIOS/UEFI (Intel VT-x/AMD-V) та активований «Windows Hypervisor» (або WSL2/Hyper-V залежно від конфігурації). Це забезпечить коректну роботу Android-емулятора без помилок під час запуску.

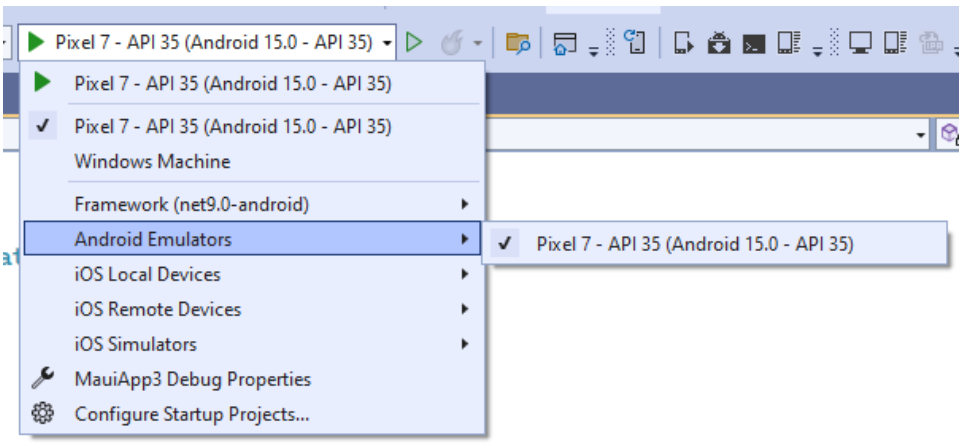


Рисунок 3.4 – Випадаюче меню з вибором Android емулятора

Спочатку потрібно відкрити рішення в Visual Studio і переконатися, що стартовим проєктом виставлено саме MAUI-проєкт (іконка у Solution Explorer має позначку .NET MAUI; за потреби – правий клік на проєкті → «Set as Startup Project»). У верхній панелі інструментів знайди селектор цільової платформи/пристрою. Для MAUI доступні Android (рис. 3.4), Windows, iOS тощо – обрати Android. Якщо список пристроїв порожній, потрібно натиснути на випадаючий список і вибрати Android Emulator, а далі – конкретний віртуальний пристрій (наприклад, Pixel 5 – Android 13). Якщо жодного немає, необхідно відкрити «Android Device Manager» («Tools» → «Android» → «Android Device Manager») і створити новий AVD, підібравши образ системи з рекомендованою ABI (x86_64) для найкращої продуктивності.

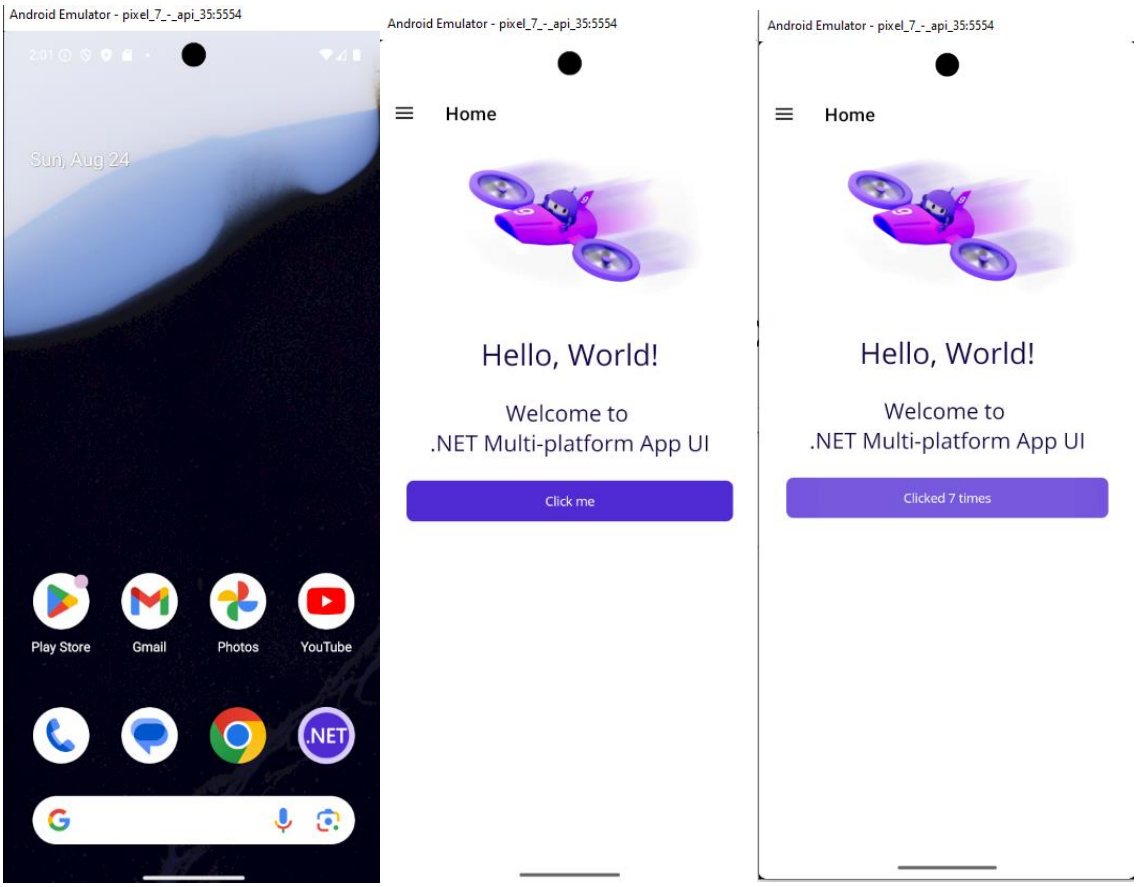


Рисунок 3.5 – Зовнішній вигляд інтерфейсу нового застосунку на .NET MAUI під час розгортання на «Android Emulator»

Перед першим запуском доцільно оновити NuGet-пакети та відновити їх: у меню «Build» виконай «Restore» або натисни правою кнопкою на рішенні і далі «Restore». Це зніме типові помилки відсутніх залежностей. Також необхідно перевірити файл csproj: цільові фреймворки мають включати net8.0-android (або відповідну версію), а «Android SDK/Platform» у тебе встановлені під цю версію. Якщо використовуються шрифти чи зображення у «Resources», необхідно переконатися, що вони реально існують, аби не зірвати збірку в порожньому шаблоні.

Далі запускаємо емульований пристрій. Можна натиснути «Play» біля обраного AVD у «Android Device Manager», або просто натиснути F5/Debug → «Start Debugging» у Visual Studio – у цьому разі IDE спробує автоматично підняти емулятор. Перший старт AVD займає кілька хвилин: ініціалізується віртуальна машина, Android завантажується до робочого столу. Далі дочекатися, поки емулятор покаже повністю завантажену систему (екран блокування/головний екран (рис. 3.5).

Після запуску емулятора натисніть F5 для збірки та деплою застосунку (або Ctrl+F5 для запуску без відлагодження). Visual Studio виконає складання для Android, згенерує .apk/.aab, підпише «debug-сертифікатом» і встановить пакет у віртуальний пристрій. У вікні «Output/Build» можна відстежити хід складання, а у «Debug Console» побачити логи рантайму. При першому запуску MAUI-шаблон, як правило, показує стартову сторінку з кнопкою-лічильником – по натисканню число збільшується, тому це простий спосіб перевірити, що все працює.

У процесі налагодження можна ставити брейкпоїнти в коді (наприклад, у обробнику натискання кнопки в «MainPage.xaml.cs»). Коли застосунок запущений у емуляторі, будь-який брейкпоїнт зупинить виконання з можливістю перегляду змінних, стеку викликів та швидкого редагування коду. Якщо потрібна гаряча перезагрузка інтерфейсу, увімкніть «Hot Reload» зміни у XAML застосуються без повного перезапуску, що пришвидшує ітерації у навіть порожньому шаблоні.

3.4.1 Поширені проблеми з запуском Android Emulator і як їх вирішити

Віртуалізація вимкнена або конфліктує з іншими гіпервізорами

Найчастіша причина, коли емультор не стартує або зависає на чорному екрані, відсутня апаратна віртуалізація (Intel VT-x/AMD-V) або її блокує інший гіпервізор. На Windows емуляторі Android x86_64 використовує Hyper-V/Windows Hypervisor Platform (WHPX), і конфлікти з VirtualBox/VMware/старим HAXM провокують збої. Рішення: увімкнути VT-x/AMD-V у BIOS/UEFI; у Windows активувати «Windows Hypervisor Platform» та Hyper-V (або, якщо використовується інший гіпервізор, навпаки – вимкнути Hyper-V/WHP). Перевірити стан можна в «Task Manager → Performance → CPU» (пункт «Virtualization: Enabled»). Після зміни налаштувань виконати повне перезавантаження.

Конфлікти драйверів графіки/рендерера

Чорний екран, «емулятор стартує, але нічого не показує», або краші під час рендерингу «OpenGL/ANGLE» часто пов’язані з драйверами GPU чи невдалим графічним бекендом. Рішення: у налаштуваннях AVD встановити «Emulated Performance» → «Graphics» = «Hardware або Automatic»; якщо не допомагає – переключити на «Software». Оновити драйвери відеокарти (особливо на ноутбуках з двома GPU) та в налаштуваннях емулятора вимкнути «Use host GPU» для проблемних систем.

Несумісні або неоновлені компоненти Android SDK

Старі версії «Android Emulator/Platform Tools/Build Tools/Platform Images» можуть викликати помилки встановлення .apk, adb-з’єднання або «emulator not found». Рішення: через «SDK Manager» оновити «Android Emulator», «Android SDK Platform-Tools» («adb»), відповідні «SDK Platforms» і «Google APIs» для цільової версії. Після оновлення перезапустити IDE та емулятор. У терміналі перевірити «*adb devices*», за потреби виконати «*adb kill-server && adb start-server*».

3.5 Запуск першої MAUI-програми на MacOS у новому проєкті

Для запуску першого проекту .NET MAUI на macOS необхідно правильно налаштувати середовище розробки. Починається це з встановлення необхідних компонентів. Перш за все, потрібно встановити останню версію пакета SDK для .NET, який включає підтримку .NET MAUI. Це можна зробити завантаженням інсталятора з офіційного сайту Microsoft або через термінал за допомогою команди «*dotnet –version»*, щоб переконатися, що встановлена потрібна версія.

Наступним кроком є встановлення Xcode – офіційного середовища розробки Apple. Xcode необхідний для збірки, запуску та налагодження додатків iOS та macOS. Його можна завантажити з App Store. Після встановлення слід відкрити Xcode, прийняти ліцензійну угоду та дозволити автоматичну установку додаткових компонентів, таких як симулятори пристроїв. Також важливо встановити інструменти командного рядка Xcode, виконавши в терміналі команду «*xcode-select – install»*.

Після підготовки базових інструментів можна встановити Visual Studio Code – полегшену IDE, яка добре підтримує .NET MAUI через спеціальне розширення. У Visual Studio Code слід встановити розширення ".NET MAUI", щоб отримати доступ до шаблонів проектів, підказок та інструментів налаштування. Після цього можна створити новий проект, використовуючи команду «*dotnet new maui -n "MyMauiApp"»* у терміналі, що створить базовий шаблон додатку.

Для збірки та запуску додатку використовується .NET CLI. Наприклад, щоб запустити додаток у симуляторі iOS, виконується команда «*dotnet build -t:Run -f net8.0-ios»* з каталогу проекту. Ця команда автоматично відновить залежності, збере проект і запустить його у симуляторі за замовчуванням. Якщо потрібно запустити додаток на фізичному пристрої, слід попередньо зареєструвати його в обліковому записі розробника Apple, створити профіль підготовки та вказати UDID пристрою в команді збірки.

Після успішного запуску можна спостерігати стандартний інтерфейс додатку .NET MAUI – сторінку з кнопкою "Click me" та лічильником натискань. Це демонструє працездатність проекту та базову взаємодію з користувачем. Подальша розробка полягає у модифікації XAML-файлів та коду C#, щоб реалізувати потрібну функціональність. Важливо пам’ятати, що для розробки під iOS та macOS обов’язково потрібен Mac, оскільки інструменти Apple працюють лише на цій платформі. Більше інформації щодо підключення пристроїв Apple та налаштування середовища Xcode можна знайти на офіційному сайті MAUI від Microsoft.

4 РОБОЧЕ ЗАВДАННЯ

Завдання 1. Проаналізувати теоретичні основи розробки кросплатформових додатків за допомогою .NET MAUI, викладені в розділі 1 навчального матеріалу. На основі отриманих знань дослідити функціональні можливості інтегрованого середовища розробки Visual Studio 2022, зокрема інструменти для створення, налагодження та тестування MAUI-додатків. Практично реалізувати всі наведені у розділі приклади, забезпечивши їх працездатність на принаймні одній цільовій платформі (Windows, Android або iOS).

Завдання 2. Розробити кросплатформовий додаток із назвою «Zavdannya2» за допомогою технології .NET MAUI. Головний екран додатка має містити текстовий елемент, вирівняний по центру вікна, який відображає рядок «Моя перша програма». Реалізацію виконати за допомогою засобів .NET MAUI із дотриманням основних принципів структурування інтерфейсу користувача.

5 КОНТРОЛЬНІ ПИТАННЯ

1 Який загальний порядок створення кросплатформового додатку у середовищі Visual Studio з використанням .NET MAUI?

2 З яких основних компонентів складається архітектура типового .NET MAUI-додатку?

3 У яких файлах розташовуються декларативні XAML-інструкції та C#-код логіки програми

4 Яке призначення файлу App.xaml у структурі .NET MAUI-проєкту?

5 Яке призначення файлу App.xaml.cs та які ключові методи він містить?

6 Що визначає файл MainPage.xaml і яку роль він відіграє у відображенні інтерфейсу додатку?

7 Де в проєкті зберігається метадані про склад і конфігурацію додатку (наприклад, список ресурсів, посилання на сторінки тощо)?

8 Яка інформація виводиться у вікні «Вивід» під час компіляції та виконання додатку?

9 Де можна знайти детальну інформацію про помилки компіляції та виконання, і як її інтерпретувати?

10 Яку послідовність дій потрібно виконати для запуску .NET MAUI-додатку на емуляторі Windows?

11 Як виконати налагодження додатку на емуляторі Android через Visual Studio?

ПРАКТИЧНА РОБОТА №2

Тема роботи: дослідження можливостей інтегрованого середовища розробки Visual Studio для створення простих кросплатформних додатків із текстовими елементами на .NET MAUI.

Мета роботи: дослідити можливості інтегрованого середовища розробки Visual Studio і отримати практичні навички створення простих кросплатформних додатків із текстовими елементами з використанням .NET Multi-platform App UI (MAUI).

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

.NET MAUI (Multi-platform App UI) – це сучасний фреймворк для створення кросплатформних застосунків на основі мови C# і платформи .NET. Його повна назва означає «багатоплатформовий користувацький інтерфейс застосунків» (Multi-platform App UI). MAUI є логічним продовженням технології Xamarin.Forms і пропонує розробникам єдиний спосіб створення програм для Android, iOS, macOS та Windows з використанням спільного коду та єдиної структури інтерфейсу.

Фреймворк визначає архітектуру застосунку, надаючи набір інструментів, компонентів і шаблонів, які спрощують створення, налагодження та підтримку складних програмних систем. .NET MAUI дозволяє розробляти мобільні та настільні програми з єдиною базою коду, завдяки чому значно скорочується час розробки та обсяг супроводу.

У .NET MAUI графічний інтерфейс користувача описується за допомогою XAML (Extensible Application Markup Language) або програмно (через C# API). Це дає змогу гнучко комбінувати логіку і візуальну частину застосунку. Компоненти інтерфейсу називаються елементами керування («controls»).

У .NET MAUI для роботи з текстовими елементами інтерфейсу використовуються такі основні класи:

- **Label** – простий елемент для відображення текстової інформації (статичний текст, заголовки, підписи до полів тощо). Приклад використання: відображення назви або підказки на екрані.
- **Entry** – однорядкове текстове поле введення даних. Застосовується для короткого тексту, наприклад, введення логіна, пароля або назви елемента. Підтримує властивості для керування типом клавіатури (наприклад, числова), довжиною рядка та поведінкою автозаміни.
- **Editor** – багаторядкове текстове поле, призначене для введення довгих текстів (коментарів, описів, повідомлень тощо). Має розширені можливості для редагування тексту, підтримує прокручування і налаштування висоти.

Ці елементи є частиною простору імен Microsoft.Maui.Controls і можуть використовуватися як у XAML, так і у коді C#. Завдяки єдиній моделі відображення, .NET MAUI автоматично підлаштовує вигляд і поведінку цих елементів під конкретну платформу.

1.1 Текстове поле **Label**

Елемент **Label** у .NET MAUI є базовим візуальним компонентом для виведення текстової інформації на екрані. Він належить до простору імен Microsoft.Maui.Controls і використовується для відображення статичного тексту, описів, підписів до інших елементів управління, а також коротких повідомлень користувачеві.

Основні характеристики елемента **Label**

Label не підтримує введення даних (на відміну від **Entry** чи **Editor**). Його основна мета – відображення тексту з можливістю налаштування вигляду, кольору, розміщення та стилю шрифту.

Клас **Label** успадковує базовий клас *View*, тому підтримує спільні властивості компонування, наприклад, HorizontalOptions, VerticalOptions, Margin, Padding тощо.

Основні властивості **Label** перелічені у таблиці 1.1.

Таблиця 1.1 – Властивості елемента **Label**

Властивість	**Опис**
Text	Основна властивість, що задає відображуваний текст. Приймає рядкове значення.
TextColor	Колір тексту. Може бути встановлений за допомогою системних кольорів або стандартного класу Colors.
FontSize	Розмір шрифту у відносних або фіксованих одиницях. Для адаптації під різні пристрої рекомендується використовувати значення з класу NamedSize.
FontAttributes	Визначає стиль шрифту: Bold, Italic, None. Може комбінуватися, наприклад: `FontAttributes.Bold
FontFamily	Назва сімейства шрифтів (напр. «OpenSans-Regular» або користувацький шрифт, доданий до проєкту).
HorizontalTextAlignment	Горизонтальне вирівнювання тексту: Start, Center, End.
VerticalTextAlignment	Вертикальне вирівнювання тексту: Start, Center, End.
LineBreakMode	Поведінка тексту, якщо він виходить за межі доступного простору (WordWrap, CharacterWrap, NoWrap, HeadTruncation, TailTruncation, MiddleTruncation).
MaxLines	Кількість рядків, які може відобразити **Label** (від 1 і більше).
Padding	Внутрішні поля всередині **Label**, особливо корисно при використанні **Label** у складних розмітках.

У .NET MAUI елемент Label підтримує рендеринг на всіх цільових платформах (Android, iOS, Windows, macOS). Зовнішній вигляд та типографіка залежать від системних налаштувань платформи, однак усі ключові властивості поведуть себе однаково. Рекомендується перевіряти вигляд шрифтів на різних платформах під час розробки, оскільки не всі системні шрифти є універсально доступними.

Приклад 1. Створення **Label** у XAML (рис. 1.1).

```
<Label
    Text="Привітання, MAUI!"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    FontSize="24"
    TextColor="DarkBlue"
    FontAttributes="Bold"
    HorizontalTextAlignment="Center"/>
```

Рисунок 1.1 – Створення **Label** у XAML

У даному прикладі *VerticalOptions* та *HorizontalOptions* визначають розташування **Label** у межах контейнера; *FontSize* – задає розмір шрифту; *TextColor* – колір тексту; *FontAttributes* – виділяє текст напівжирним накресленням.

Приклад 2. Створення **Label** у C#. На рнсунку 1.2 ви можете бачити приклад реалізації текстового елемента **Label** на мові програмування C#. Цей спосіб є зручним, якщо потрібно динамічно створювати або змінювати **Label** під час виконання програми.

```
using Microsoft.Maui.Controls;

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();

        var lbl = new Label
        {
            Text = "Привітання, MAUI!",
            VerticalOptions = LayoutOptions.Center,
            HorizontalOptions = LayoutOptions.Center,
            FontSize = 24,
            TextColor = Colors.DarkBlue,
            FontAttributes = FontAttributes.Bold,
            HorizontalTextAlignment = TextAlignment.Center
        };

        Content = lbl;
    }
}
```

Рисунок 1.2 – Створення **Label** у C#

1.2 Текстове поле **Entry**

Елемент **Entry** в .NET MAUI є фундаментальним компонентом для створення однорядкових текстових полів, які призначені для введення користувачами короткої текстової інформації. Це аналог стандартного елементу **TextBox** у інших платформах, але адаптований до кросплатформенного середовища, що дозволяє створювати однакові інтерфейси для Android, iOS, Windows та macOS.

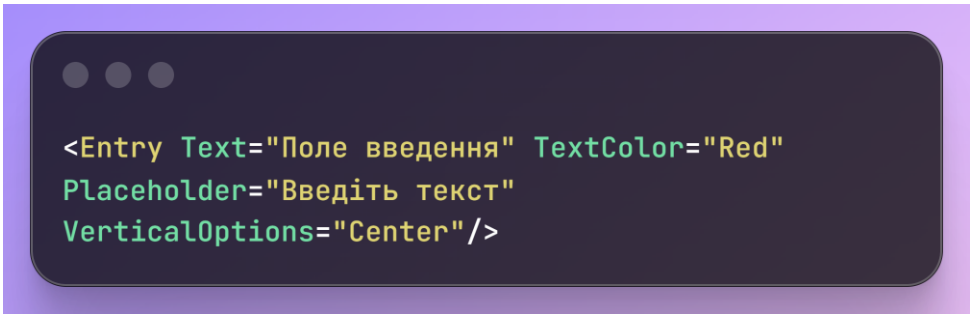
Основні властивості елемента **Entry**:

- *Text*. Ця властивість зберігає й відображає поточний введений користувачем текст. Вона є основною для отримання або налаштування значення поля. Наприклад, при початковому створенні елемента її можна ініціалізувати певним рядком або отримувати значення при натисканні кнопки.

- *TextColor*. Визначає колір тексту, який вводиться у поле. Це корисно для створення контрастних або стилізованих інтерфейсів згідно з дизайном застосунку. Встановити його можна за допомогою стандартних кольорів або створюючи власні.
- *Placeholder*. Це текстовий індикатор, який допомагає користувачу зрозуміти, яку інформацію потрібно ввести у поле. Він відображається, поки користувач не почне вводити дані, і зникає одразу після початку набору.
- *IsPassword*. Булева властивість, яка визначає, чи є це поле для введення пароля. При значенні true всі введені символи автоматично замінюються на зірочки або будь-який інший прихований символ в залежності від платформи, що забезпечує безпеку введених даних.
- *Keyboard*. Дозволяє керувати типом клавіатури, яка відкривається під час редагування поля. Властивість використовує перерахування Keyboard та може приймати такі значення: *Default* – стандартна клавіатура платформи; *Text* – текстова клавіатура з буквами та символами; *Chat* – клавіатура для швидкого набору повідомлень; *Url* – клавіатура для адресів вебсайтів; *Email* – клавіатура для введення email-адрес; *Telephone* – клавіатура для телефонних номерів; *Numeric* – цифрова клавіатура для введення чисел. Наприклад, якщо необхідно створити поле для введення телефону, то можна задати *Keyboard = Keyboard.Telephone*, і при фокусуванні користувач отримає клавіатуру з цифрами та зірочками для зручнішого набору.

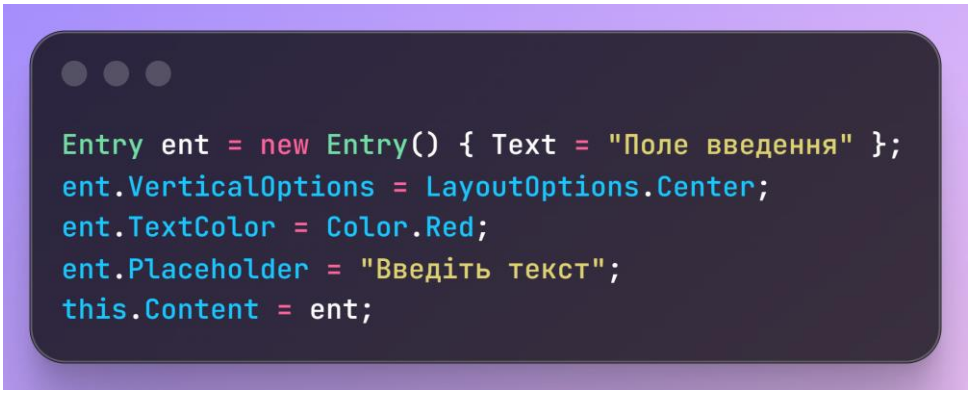
Події **Entry**: *TextChanged*. Виникає щоразу, коли у полі змінюється текст. Це корисно для динамічного реагування, наприклад, при валідації введених даних або під час автодоповнення; *Completed*. Спричинена завершенням редагування, наприклад, натисканням клавіші "Enter" або втратою фокусу. Це сигналізує про завершення вводу і може використовуватися для обробки введених даних.

Приклад створення поля **Entry** показаний на рисунку 1.3 на мові XAML та на мові C# (рис. 1.4).



```xaml
<Entry Text="Поле введення" TextColor="Red"
Placeholder="Введіть текст"
VerticalOptions="Center"/>
```

Рисунок 1.3 – Створення елементу **Entry** у XAML



```csharp
Entry ent = new Entry() { Text = "Поле введення" };
ent.VerticalOptions = LayoutOptions.Center;
ent.TextColor = Color.Red;
ent.Placeholder = "Введіть текст";
this.Content = ent;
```

Рисунок 1.4 – Створення елементу **Entry** у C#

Такий спосіб дає можливість створювати і налаштовувати елементи у коді, що корисно при динамічному створенні інтерфейсу або необхідності більш гнучкого управління.

1.3 Текстове поле **Editor**

Елемент **Editor** у технології .NET MAUI призначений для введення та редагування багаторядкового тексту. На відміну від елемента Entry, який використовується для введення коротких однорядкових даних (наприклад, логінів або назв), **Editor** забезпечує розширену функціональність для введення довших текстів – коментарів, описів чи нотаток. Він автоматично підтримує перенос тексту, дозволяє вводити довільну кількість рядків і динамічно змінює висоту вмісту.

Завдяки своїй універсальності елемент **Editor** часто використовується у формах, анкетах або будь-яких інтерфейсах, де користувачу необхідно ввести описову інформацію. Компонент є частиною бібліотеки .NET MAUI **Controls** і працює на всіх підтримуваних платформах (Android, iOS, macOS, Windows) з урахуванням особливостей нативного відображення кожної з них.

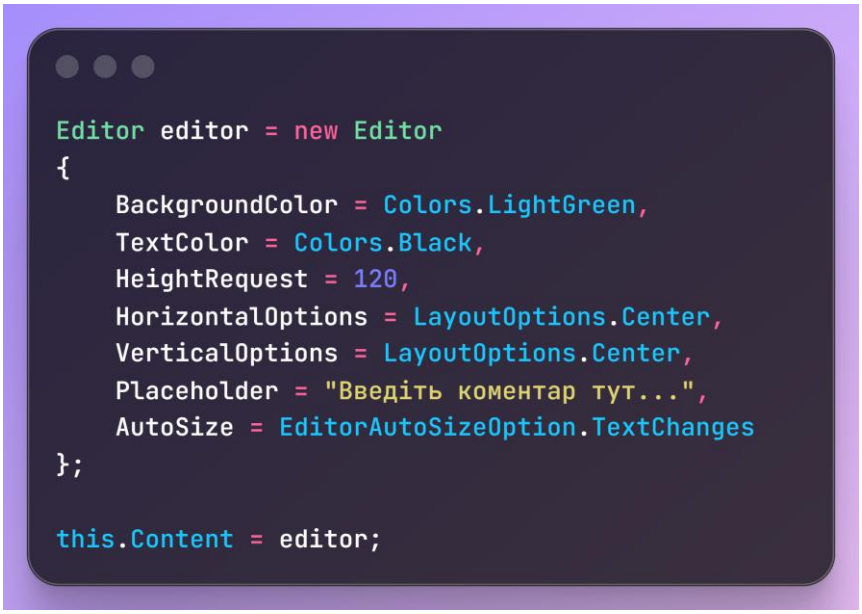
Основні властивості елементу **Editor** має подібні до елементу **Entry**, але з додатковими параметрами для багаторядкового введення: *Text* – текстовий вміст поля; *Placeholder* – текст-заповнювач, який з'являється, коли поле порожнє; *TextColor* – колір тексту користувача; *BackgroundColor* – колір фону елемента; *FontSize*, *FontAttributes*, *FontFamily* – налаштування шрифту; *IsReadOnly* – визначає, чи може користувач змінювати текст; *Keyboard* – встановлює тип клавіатури (текстова, числова, електронна пошта тощо); *AutoSize* – визначає, чи змінюватиметься висота поля залежно від кількості рядків тексту (доступно з .NET 8 MAUI).

Елемент **Editor** можна створити двома основними способами: за допомогою мови XAML (рис. 1.5) або програмно – через C# (рис. 1.6). Обидва підходи дають однаковий результат, але XAML зазвичай використовують у статичних інтерфейсах, тоді як програмне створення зручне для динамічних компонентів.

```
<Editor
    BackgroundColor="LightGreen"
    TextColor="Black"
    HeightRequest="120"
    HorizontalOptions="Center"
    VerticalOptions="Center"
    Placeholder="Введіть коментар тут..."
    AutoSize="TextChanges" />
```

Рисунок 1.5 – Створення елементу **Editor** у XAML

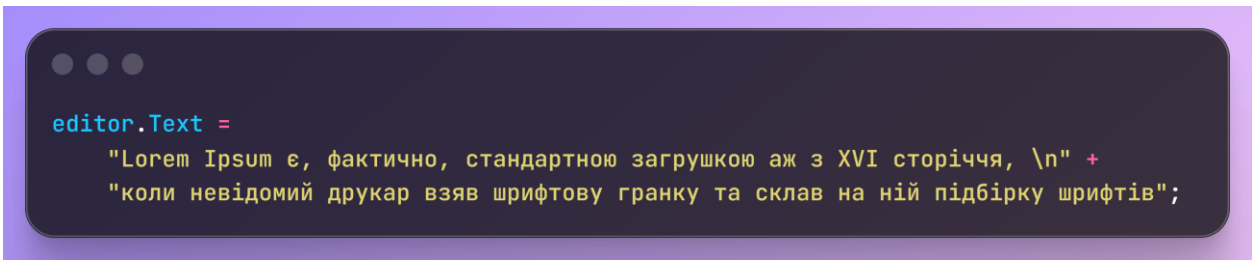
У цьому прикладі елемент матиме зелений фон, чорний текст і автоматично змінюватиме висоту при наборі нового тексту.



```
Editor editor = new Editor
{
    BackgroundColor = Colors.LightGreen,
    TextColor = Colors.Black,
    HeightRequest = 120,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    Placeholder = "Введіть коментар тут...",
    AutoSize = EditorAutoSizeOption.TextChanges
};

this.Content = editor;
```

Рисунок 1.6 – Створення елементу **Editor** у C#

Для перенесення тексту на новий рядок (рис. 1.7) у властивості *Text* використовується символ переведення рядка **\n**. При виконанні програми цей текст буде відображено у двох рядках. У XAML для цього можна застосувати послідовність **
**



```
editor.Text =
    "Lorem Ipsum є, фактично, стандартною загрушкою аж з XVI сторіччя, \n" +
    "коли невідомий друкар взяв шрифтову гранку та склав на ній підбірку шрифтів";
```

Рисунок 1.7 – Перенесення тексту на новий рядок **Editor** у C#

1.4 Розміщення кількох елементів

У технології MAUI (Multi-platform App UI) візуальний інтерфейс користувача, як і у Xamarin.Forms, складається з так званих сторінок (**Page**),

які є базовими об’єктами представлення. Сторінка є екранною областю, що займає увесь простір вікна додатку на пристрої. Вона відповідає класу **Page** і слугує контейнером для відображення вмісту програми. Програма може містити одну або декілька сторінок, переважно з переходом між ними через навігаційний стек.

У MAUI сторінка приймає як вміст один основний елемент компонування (layout), який у свою чергу може вміщувати довільну кількість візуальних елементів (контролів). Це пов’язано з властивістю *Content* класу *ContentPage*, яка обмежена одним елементом типу *View*, тобто одною розміткою або контейнером. За замовчуванням *ContentPage* може містити простий елемент, наприклад **Label**, але для розміщення одночасно декількох елементів потрібні спеціалізовані контейнери для компонування, такі як **StackLayout**, **AbsoluteLayout**, **RelativeLayout** або **Grid**.

Для організації та розташування кількох елементів у MAUI найчастіше використовується контейнер **StackLayout**. Це один із базових видів layout-контейнерів, який формує внутрішній вміст у вигляді стеку – послідовного розміщення дочірніх елементів у вертикальному або горизонтальному напрямку. **StackLayout** автоматично розташовує свої дочірні елементи підряд, що дуже зручно для формування списків, груп кнопок, текстових блоків і інших стандартних інтерфейсних компонентів.

Властивість *Children* у кожного елемента компонування (layout) містить колекцію вкладених елементів, які створюють структуру ієрархії візуальних об’єктів. Ця колекція підтримує додавання, видалення або заміну дочірніх елементів програмним способом. У MAUI так само, як і у Xamarin.Forms, можна описувати ієрархію інтерфейсу як у декларативній мові розмітки XAML, так і у імперативному коді на C#.

Приклад використання **StackLayout** у XAML зображений на рисунку 1.8, а на мові програмування C# – на рисунку 1.9.

```xaml
<StackLayout x:Name="stackLayout" Orientation="Vertical">
  <StackLayout.Children>
    <Label Text=" Перша мітка]" />
    <Label Text=" Друга мітка]" />
  </StackLayout.Children>
</StackLayout>
```

Рисунок 1.8 – Приклад використання **StackLayout** у XAML

```csharp
public class MainPage : ContentPage
{
  public MainPage()
  {
    Label lbl1 = new Label { Text = "Перша мітка]" };
    Label lbl2 = new Label { Text = "Друга мітка]" };

    StackLayout stackLayout = new StackLayout
    {
      Orientation = StackOrientation.Vertical,
      Children = { lbl1, lbl2 }
    };

    this.Content = stackLayout;
  }
}
```

Рисунок 1.9 – Приклад використання **StackLayout** на C#

1.5 Позиціонування та форматування елементів на сторінці

1.5.1 Вирівнювання тексту на сторінці

У MAUI всі візуальні компоненти інтерфейсу користувача успадковуються від базового класу **View**. Цей клас містить дві основні властивості – *HorizontalOptions* та *VerticalOptions*, які відповідають за вирівнювання елемента відповідно по горизонталі та вертикалі. Кожна з цих властивостей приймає значення із набору констант перерахування

LayoutOptions. Ці константи визначають різні способи розміщення елементів у контейнері. Значення перерахування LayoutOptions включають: *Start* – вирівнювання по лівому краю для горизонталі або по верху для вертикалі; *Center* – центроване розміщення елемента в межах доступного простору; *End* – вирівнювання по правому краю (горизонтально) або знизу (вертикально); *Fill* – елемент розтягується, заповнюючи весь простір контейнера; *StartAndExpand*, *CenterAndExpand*, *EndAndExpand*, *FillAndExpand* – варіанти з аналогічним вирівнюванням, але з додатковою властивістю розтягування, яка дозволяє елементу займати додатковий доступний простір у контейнері.

Це вирівнювання можна задавати як через розмітку XAML (рис. 1.10), так і через C# код (рис. 1.11). Наприклад, у XAML можна визначити елемент **Label** з вирівнюванням по центру наступним чином.

```
<Label Text="Приклад" VerticalOptions="Center" HorizontalOptions="Center"/>
```

Рисунок 1.10 – Вирівнювання елемента **Label** по центру у XAML

```
Label lbl = new Label() { Text = "Приклад" };
lbl.VerticalOptions = LayoutOptions.Center;
lbl.HorizontalOptions = LayoutOptions.Center;
```

Рисунок 1.11 – Вирівнювання елемента **Label** по центру у C#

1.5.2 Вирівнювання тексту всередині елементів

Окремо від розташування самих елементів, часто потрібно налаштовувати вирівнювання тексту, який вони містять. Для цього

застосовуються властивості *HorizontalTextAlignment* та *VerticalTextAlignment*, які визначають горизонтальне та вертикальне вирівнювання тексту всередині елемента. Ці властивості отримують значення з переліку **TextAlignment**: *Start* – текст вирівнюється по лівому краю або по верху; *Center* – текст центрується всередині елемента; *End* – текст вирівнюється по правому краю або по низу.

Приклад на мові XAML та на мові C# зображені на рисунку 1.12 та рисунку 1.13 відповідно.

Рисунок 1.12 – Вирівнювання елемента **Label** по центру на мові XAML

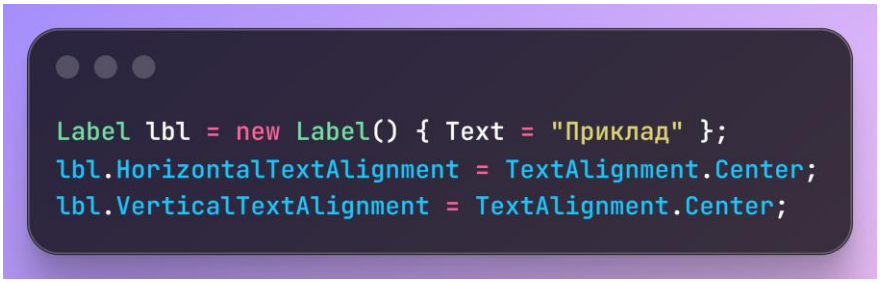


Рисунок 1.13 – Вирівнювання елемента **Label** по центру у C#

1.5.3 Робота з кольором

Колір тексту та фону елементів задається властивостями *TextColor* та *BackgroundColor*, які приймають значення типу **Color**. Для установки кольорів можна використовувати як вбудовані колірні константи MAUI (наприклад, Color.Blue, Color.Yellow), так і створювати кольори з компонентів RGB або RGBA.

Приклад на мові XAML та на мові C# зображені на рисунку 1.14 та рисунку 1.15 відповідно.

```
<Label Text="Приклад"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Center"
BackgroundColor="Blue" TextColor="Yellow" />
```

Рисунок 1.14 – Встановлення кольору елементу та фону **Label** на мові XAML

```
Label lbl = new Label() { Text = "Приклад" };
lbl.HorizontalTextAlignment = TextAlignment.Center;
lbl.VerticalTextAlignment = TextAlignment.Center;
lbl.BackgroundColor = Color.Blue;
lbl.TextColor = Color.Yellow;
```

Рисунок 1.15 – Встановлення кольору елементу та фону **Label** по центру у C#

Для більш тонкого налаштування кольору можна створювати об’єкти **Color**, передаючи компоненти червоного, зеленого та синього кольорів у діапазоні від 0.0 до 1.0, а також параметр прозорості. Наприклад: *lbl.BackgroundColor = new Color(0.9, 0.9, 0.8)*. Також є статичні методи для зручного створення кольорів:

- *Color.FromRgb(double r, double g, double b)* – компоненти червоного, зеленого та синього (0–1);
- *Color.FromRgb(int r, int g, int b)* – компоненти як цілі числа від 0 до 255;
- *Color.FromRgba(double r, double g, double b, double a)* – додає прозорість (0.0–1.0);
- *Color.FromRgba(int r, int g, int b, int a)* – прозорість у діапазоні 0–255;
- *Color.FromHsla(double h, double s, double l, double a)* – задає колір через тон (hue), насиченість (saturation), яскравість (luminosity) та прозорість.

Наприклад: *lbl.TextColor = Color.FromRgba(255, 0, 0, 160);* Або на XAML: *<Label Text="Приклад" BackgroundColor="#a7a7aa" TextColor="Red" />*

1.5.4 Стилізація тексту

MAUI надає властивості для налаштування зовнішнього вигляду тексту:

- *FontFamily* – задає шрифт, наприклад, «Arial»;
- *FontSize* – розмір шрифту, задається числовим значенням або іменованим розміром (наприклад, Large);
- *FontAttributes* – дозволяє застосувати візуальні ефекти, такі як напівжирний (Bold) або курсив (Italic).

Приклади встановлення шрифтів у XAML:

<Label Text="Приклад" FontFamily="Arial" /> та на C# (рис. 1.16).

Рисунок 1.16 – Встановлення типу шрифту Arial у C#

Встановлення розміру шрифту у XAML: *<Label Text="Приклад" FontSize="24"/>* У C# можна встановити розмір шрифту як конкретне число або скористатися методом Device.GetNamedSize(), який приймає розмір із переліку *NamedSize* та тип елемента (зазвичай **Label**): *lbl.FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label));*

1.5.5 Форматування тексту в MAUI

У платформі MAUI елемент **Label** призначений для відображення тексту у інтерфейсі користувача. Найпростіша властивість для задання тексту – це властивість *Text*, яка дозволяє встановити простий рядок тексту без форматування. Однак, коли є потреба у складнішому оформленні тексту, наприклад, змінити розмір, колір, стиль окремих частин тексту всередині одного **Label**, застосовується властивість *FormattedText*.

Властивість *FormattedText* приймає об’єкт типу **FormattedString**. Цей об’єкт інкапсулює колекцію елементів **Span**, кожен з яких відповідає за невеличкий фрагмент тексту зі своїм власним набором стилів. Для оформлення **Span** можна використовувати ті ж властивості, що й для **Label**, зокрема *FontSize*, *FontAttributes* (напівжирний, курсив), *ForegroundColor* тощо. Таке форматування дає змогу створювати комбінований текст, де в одному рядку можуть поєднуватися різні шрифтові стилі і кольори.

Наприклад, у розмітці XAML можна задати **Label** зі складним форматуванням наступним чином (рисунок 1.17).

```
<Label VerticalTextAlignment="Center" HorizontalTextAlignment="Center">
  <Label.FormattedText>
    <FormattedString>
      <Span Text="Сьогодні " FontSize="Large" />
      <Span Text="хороша" FontAttributes="Bold" />
      <Span Text="погода!" ForegroundColor="Red" />
    </FormattedString>
  </Label.FormattedText>
</Label>
```

Рисунок 1.17 – Реалізація складного форматування **Label** у XAML

У коді на C# (рис. 1.18) це еквівалентно створенню об’єкту *FormattedString* та додаванню у нього **Span** з потрібними властивостями, а потім присвоєнню цього об’єкту властивості *FormattedText* **Label**.

1.5.6 Відступи в MAUI

Для управління просторовими відступами у елементах інтерфейсу MAUI використовує дві ключові властивості – *Margin* та *Padding*. Ці властивості визначають поділ простору навколо і всередині елементів.

Margin – це зовнішній відступ, який встановлює простір між межами елемента і сусідніми елементами або контейнером. Інакше кажучи, *Margin* «відсуває» елемент від інших елементів або границь батьківського контейнера.

Padding – це внутрішній відступ, який задає відстань між межами елемента і його внутрішнім вмістом, наприклад, відступ тексту від країв **Label**.

```
Label lbl = new Label();
this.Content = lbl;
FormattedString formattedString = new FormattedString();
formattedString.Spans.Add(new Span
{
    Text = "Сьогодні ",
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))
});
formattedString.Spans.Add(new Span
{
    Text = "хороша",
    FontAttributes = FontAttributes.Bold
});
formattedString.Spans.Add(new Span
{
    Text = "погода!",
    ForegroundColor = Color.Red
});
lbl.FormattedText = formattedString;
lbl.VerticalTextAlignment = TextAlignment.Center;
lbl.HorizontalTextAlignment = TextAlignment.Center;
```

Рисунок 1.18 – Реалізація складного форматування **Label** у C#

Обидві властивості застосовують значення типу **Thickness**, що представляє собою чотиристоронній відступ із параметрами: *Left*, *Top*, *Right*, Bottom. Враховуючи це, можна задавати відступи кількома способами:

- одинарне значення – однаковий відступ з усіх сторін.
- два значення – перше для горизонтальних відступів (ліворуч і праворуч), друге для вертикальних (зверху та знизу).
- чотири значення – окремі відступи для кожної сторони (ліворуч, зверху, праворуч, знизу).

Розглянемо приклади встановлення відступів у XAML (рис. 1.19).

```xaml
<StackLayout Padding="60">
  <BoxView Color="Blue" Margin="50" HeightRequest="100" />
  <BoxView Color="Red" Margin="50" HeightRequest="100" />
</StackLayout>

<StackLayout Padding="0,20,0,0">
  <BoxView Color="Green" Margin="20"/>
  <BoxView Color="Blue" Margin="10,15"/>
  <BoxView Color="Red" Margin="0,20,15,5"/>
</StackLayout>
```

Рисунок 1.19 – Встановлення внутрішніх і зовнішніх відступів **StackLayout** у XAML

(перший варіант із відступами по 60 одиниць з обох боків і другий варіант без відступів зверху, знизу та ліворуч і з відступом у 20 одиниць праворуч)

Подібне задавання відступів можна виконати і у коді на C#:

```csharp
var stackLayout = new StackLayout
{
    Padding = new Thickness(0, 20, 0, 0),
    Children = {
        new BoxView { Color = Color.Green, Margin = new Thickness(20) },
        new BoxView { Color = Color.Blue, Margin = new Thickness(10, 15) },
        new BoxView { Color = Color.Red, Margin = new Thickness(0, 20, 15, 5) }
    }
};

Content = stackLayout;
```

Рисунок 1.20 – Встановлення зовнішніх і внутрішніх відступів **StackLayout** у C#

2 РОБОЧЕ ЗАВДАННЯ

Завдання 1. Проаналізувати теоретичні основи розробки кросплатформових додатків за допомогою .NET MAUI, викладені в розділі 1 навчального матеріалу. На основі отриманих знань дослідити функціональні можливості інтегрованого середовища розробки Visual Studio, зокрема

інструменти для створення текстових елементів у MAUI-додатках. Практично реалізувати всі наведені у розділі приклади, забезпечивши їх працездатність на принаймні одній цільовій платформі (Windows, Android або iOS).

Завдання 2. Розробити кросплатформовий додаток за допомогою технології .NET MAUI. Головний екран додатка має містити текстовий елемент **Label**, **Entry** та **Editor** із різним форматуванням. Реалізацію виконати за допомогою засобів .NET MAUI із дотриманням основних принципів структурування інтерфейсу користувача.

3 КОНТРОЛЬНІ ПИТАННЯ

1. Яке призначення елемента **Label** у додатках .NET MAUI?
2. Які основні властивості визначають поведінку та відображення елемента **Label**?
3. Для чого використовується елемент **Entry** у графічному інтерфейсі користувача?
4. Які ключові властивості підтримує елемент **Entry**?
5. У яких випадках доцільно використовувати елемент **Editor**?
6. Які основні властивості характеризують елемент **Editor**?
7. Які елементи компонування застосовуються під час побудови інтерфейсу користувача в .NET MAUI?
8. Назвіть основні механізми розташування елементів інтерфейсу.
9. Які властивості визначають вирівнювання візуальних елементів на сторінці?
10. Які властивості відповідають за вирівнювання текстового вмісту?
11. Які властивості використовуються для задання кольорової схеми елементів?
12. Які властивості застосовуються для стилізації текстових елементів?
13. Які властивості впливають на форматування тексту в інтерфейсі?

ПРАКТИЧНА РОБОТА №3

Тема роботи: дослідження можливостей Visual Studio та .NET MAUI: інтерактивні елементи й обробка подій

Мета роботи: дослідити можливості інтегрованого середовища розробки Visual Studio і отримати практичні навички створення простих кросплатформних додатків з використанням .NET Multi-platform App UI (MAUI) з інтерактивними елементами: Slider, Stepper, Switch, Button.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Елементи інтерфейсу користувача (UI Elements), які в MAUI ще називають елементами управління чи віджетами – це базові графічні компоненти, призначені для взаємодії користувача з програмою. Всі ці компоненти мають стандартний вигляд, функціональність та забезпечують уніфікований досвід користування незалежно від операційної системи (Windows, Android, iOS, macOS). MAUI реалізує платформонезалежний набір елементів, які автоматично адаптуються під стиль і поведінку цільової ОС.

Елементи інтерфейсу відіграють ключову роль у зручності та ефективності інтерфейсу, оскільки дозволяють розробнику швидко створювати UI, що легко підтримується та розширюється. У MAUI всі стандартні контролі реалізовані як об'єкти .NET і підтримують розширення та налаштування їх властивостей через XAML або код на C#.

1.1 Елемент **Button**

Елемент **Button** у MAUI є графічною кнопкою, яка надає користувачу можливість ініціювати певну дію негайно після натискання. Він є одним із найпоширеніших елементів управління та часто використовується для підтвердження вибору, запуску процесів або переходу між екранами. **Button** має набір властивостей, які задають його вигляд та роботу: *BorderColor* – колір рамки кнопки. Дана властивість дозволяє зробити кнопку більш помітною або

стильнішою. *BorderRadius* – радіус закруглення кутів рамки кнопки. Завдяки цьому можна створити кнопки з плавними заокругленими краями; *BorderWidth* – товщина рамки навколо кнопки, вимірюється у пікселях або відповідних одиницях пристрою; *FontAttributes* – атрибути шрифту, які визначають стиль тексту: жирний (Bold), курсив (Italic) або без стилю (None); *FontFamily* – сімейство шрифтів, яке задає типографіку тексту на кнопці; *FontSize* – розмір шрифту, задається цілим числом у одиницях пристрою, що дозволяє адаптувати текст до різних розмірів екрану; *Text* – текст, який відображається на кнопці. Його можна динамічно змінювати під час роботи додатку; *TextColor* – колір тексту, що забезпечує можливість гарного контрасту й стилізації; *Image* – властивість для додавання зображення на кнопку, що розширює її функціонал і естетику.

Головною подією, яка забезпечує взаємодію користувача з кнопкою, є подія *Clicked*. Вона спрацьовує у момент натискання на кнопку, викликаючи відповідний обробник, де можна описати логіку дій.

У .NET MAUI існують два основних способи створення і розміщення кнопки на сторінці: на мові XAML (рис. 1.1) та на мові C# (рис. 1.2).

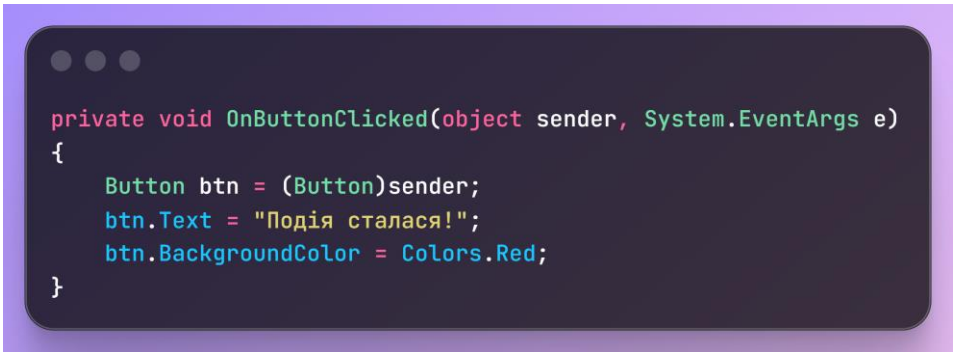
```xaml
<Button Text="Кнопка" VerticalOptions="Center" HorizontalOptions="Center"/>
```

Рисунок 1.1 – Створення елементу **Button** на XAML

```csharp
Button btn = new Button() { Text = "Кнопка" };
btn.VerticalOptions = LayoutOptions.Center;
btn.HorizontalOptions = LayoutOptions.Center;
this.Content = btn;
```

Рисунок 1.2 – Створення елементу **Button** на C#

Щоб реагувати на натискання кнопки, треба додати обробник події. Це можна зробити у XAML-файлі: *<Button Text="Кнопка" VerticalOptions="Center" Clicked="OnButtonClicked"/>* або у C#-коді: *btn.Clicked += OnButtonClicked*;. Обробник події – це метод, який виконується при натисканні кнопки. Він приймає два параметри: джерело події (об'єкт типу object) та аргументи події (*System.EventArgs*), що можуть містити додаткову інформацію (рис. 1.3).



```
private void OnButtonClicked(object sender, System.EventArgs e)
{
    Button btn = (Button)sender;
    btn.Text = "Подія сталася!";
    btn.BackgroundColor = Colors.Red;
}
```

Рисунок 1.3 – Реалізація методу *OnButtonClicked* у C#

Цей код змінює текст кнопки та її фон, показуючи користувачу, що подія спрацювала. Обробник за синтаксисом і семантикою подібний до стандартних обробників у технології Windows Forms, що робить його зрозумілим для розробників з досвідом у традиційних платформах.

1.2 Елемент **Switch**

Елемент **Switch** у .NET MAUI є інтуїтивно зрозумілим візуальним елементом управління у вигляді перемикача, який дозволяє користувачу вибирати один із двох станів: увімкнено або вимкнено. Цей елемент часто використовується у формах, налаштуваннях програм чи будь-яких інтерфейсах, де необхідно швидко і легко вмикати або вимикати певну функцію.

Основна властивість елемента **Switch** – це *IsToggled*, яка зберігає логічне булеве значення, що показує стан перемикача. Коли *IsToggled* має значення true, це означає, що **Switch** увімкнено; коли false – вимкнено. За замовчуванням це значення дорівнює false, тобто елемент вимкнений. Через цю властивість можна як програмно встановлювати стан перемикача, так і отримувати його поточний стан у коді.

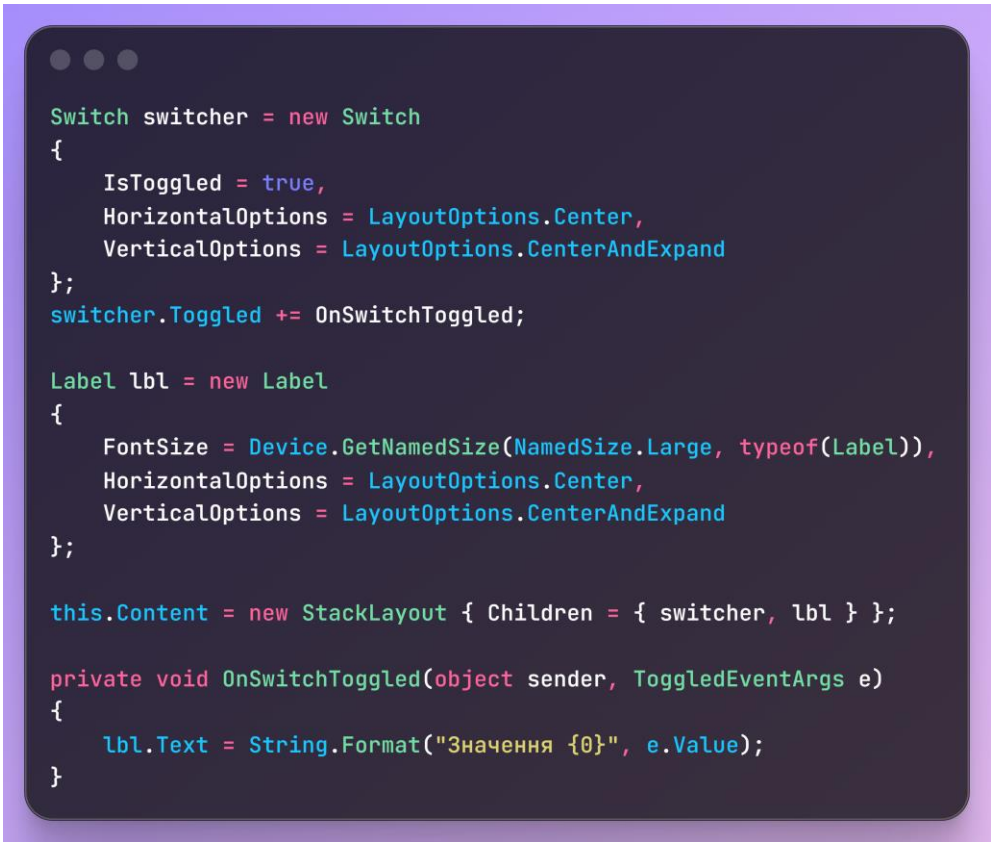
Для реагування на зміну стану **Switch** існує подія *Toggled*. Вона піднімається завжди, коли користувач змінює положення перемикача або коли стан змінюється програмно через присвоєння властивості *IsToggled*. Обробник події *Toggled* отримує об'єкт типу *ToggledEventArgs*, в якому міститься властивість *Value*, що відображає новий стан **Switch**. Ця подія є ключовою для реалізації логіки зміни поведінки додатку залежно від увімкнення чи вимкнення **Switch**.

Для зручності та кращої інтеграції в інтерфейс, **Switch** підтримує зміну зовнішнього вигляду залежно від стану. Зокрема, можна визначити кольори для "положення увімкнено" (*OnColor*) та "положення вимкнено" (*OffColor*), а також колір самого "повзунка" (*ThumbColor*). Ці властивості дозволяють стилізувати перемикач відповідно до дизайну вашого додатку. Приклад використання **Switch** у XAML показаний на рисунку 1.4.

```
<StackLayout>
    <Switch IsToggled="true"
            VerticalOptions="CenterAndExpand"
            HorizontalOptions="Center"
            Toggled="OnSwitchToggled" />
    <Label x:Name="lbl"
           FontSize="Large"
           HorizontalOptions="Center"
           VerticalOptions="CenterAndExpand" />
</StackLayout>
```

Рисунок 1.4 – Реалізація елементу **Switch** у XAML

Цей приклад створює **Switch**, який за замовчуванням увімкнений (*IsToggled*="true"), розміщений по центру. Подія *Toggled* пов’язана з методом обробки зміни стану. Поруч розміщено **Label** для відображення тексту. Аналогічний код на C# зображений на рисунку 1.5.



```
Switch switcher = new Switch
{
    IsToggled = true,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.CenterAndExpand
};
switcher.Toggled += OnSwitchToggled;

Label lbl = new Label
{
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.CenterAndExpand
};

this.Content = new StackLayout { Children = { switcher, lbl } };

private void OnSwitchToggled(object sender, ToggledEventArgs e)
{
    lbl.Text = String.Format("Значення {0}", e.Value);
}
```

Рисунок 1.5 – Реалізація елементу **Switch** у C#

У цьому коді створюється об’єкт **Switch** з початковим увімкненим станом. До події *Toggled* виконується спільний метод *OnSwitchToggled*, який оновлює текст у **Label**, показуючи, чи увімкнений елемент.

1.3 Елемент **Picker**

Елемент **Picker** у технології .NET MAUI використовується для створення інтерактивного елемента керування, який дозволяє користувачу вибрати одне значення із заздалегідь визначеного списку. За своєю суттю,

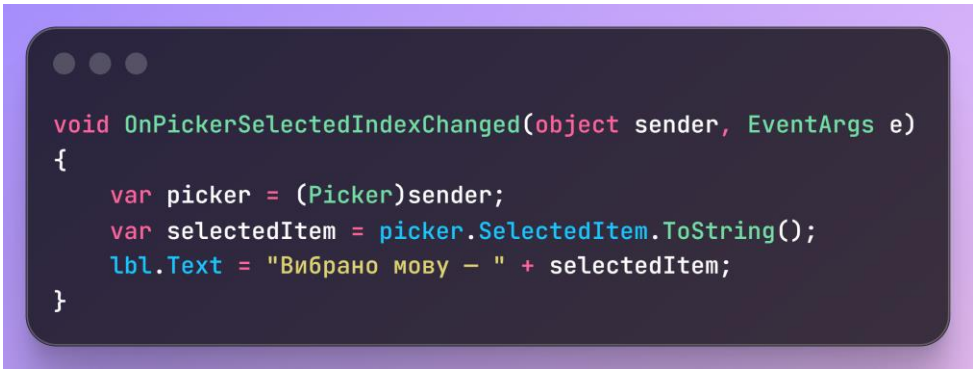
Picker нагадує стандартний елемент «випадаючий список» у настільних застосунках. Він забезпечує зручний спосіб взаємодії користувача з фіксованими множинами даних, такими як перелік мов програмування, категорій чи одиниць вимірювання. При запуску програми елемент **Picker** відображається у вигляді текстового поля з коротким описом або заголовком. Коли користувач торкається цього поля (на мобільному пристрої) або клацає його (на комп'ютері), відкривається список усіх доступних варіантів для вибору. Після вибору одного зі значень, воно автоматично відображається у полі **Picker** і може бути оброблене програмно.

З технічної точки зору, елемент **Picker** належить до простих селективних елементів введення, які не потребують складних механізмів зв'язування з базами даних. Проте за допомогою механізму **Data Binding** у MAUI, список елементів або вибране значення можна прив'язати до властивостей моделі даних (ViewModel), що робить **Picker** гнучким інструментом для застосунків за архітектурою MVVM.

Найважливіші властивості, що визначають поведінку елемента **Picker**, такі: *Items* – колекція рядків, які утворюють список вибору. Це властивість використовується для додавання всіх варіантів, доступних користувачу; *SelectedIndex* – індекс вибраного елемента у колекції. Якщо користувач ще не зробив вибір, значення дорівнює одиниці. Можна як отримати індекс, так і задати його програмно; *SelectedItem* – об'єкт, що представляє безпосередньо обране значення зі списку (альтернатива використанню SelectedIndex); *Title* – текст-заголовок, який відображається у полі до вибору елемента. Часто використовується для коротких підказок користувачу, наприклад: "Оберіть мову програмування"; *TextColor* – колір тексту вибраного значення; *CharacterSpacing, FontAttributes, FontSize, FontFamily* – параметри, що визначають візуальні характеристики текстового відображення.

Основна подія, пов'язана з **Picker –** *SelectedIndexChanged*. Вона спрацьовує щоразу, коли користувач обирає новий елемент зі списку. Ця подія використовується для виконання динамічних дій у програмі – наприклад,

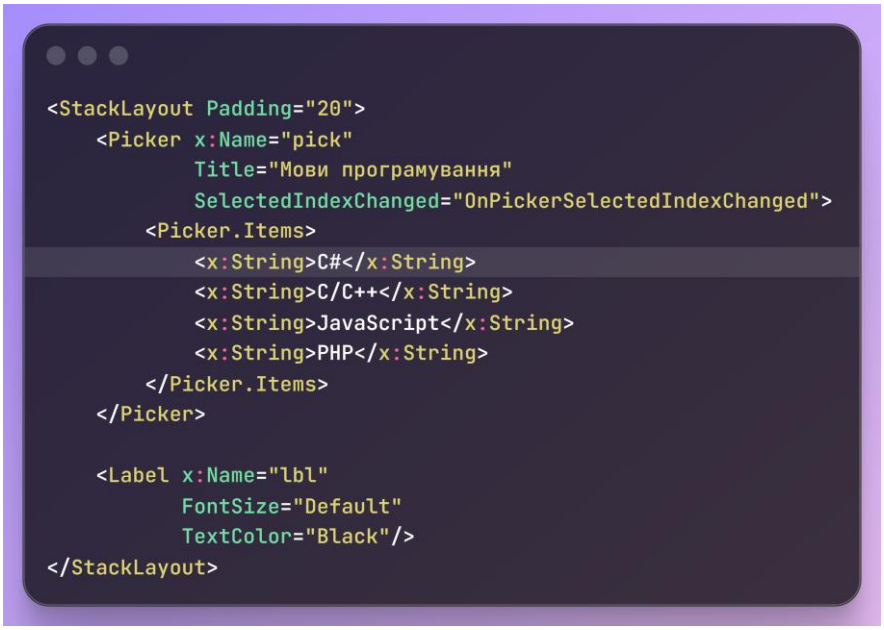
показу додаткової інформації або оновлення інших елементів інтерфейсу залежно від вибору. Приклад стандартного обробника подій зображений на рисунку 1.6.



```
void OnPickerSelectedIndexChanged(object sender, EventArgs e)
{
    var picker = (Picker)sender;
    var selectedItem = picker.SelectedItem.ToString();
    lbl.Text = "Вибрано мову – " + selectedItem;
}
```

Рисунок 1.6 – Реалізація обробника елементу **Picker** у C#

Приклад використання **Picker** у XAML показаний на рисунку 1.7. У наведеному прикладі контейнер **StackLayout** використовується для вертикального розміщення елементів, а **Picker** ініціалізується безпосередньо в XAML із вбудованим списком мов програмування. Після вибору нового значення з випадаючого списку, викликається метод *OnPickerSelectedIndexChanged*, який змінює текст мітки **Label**. На рисунку 1.8 показаний варіант створення елементу **Picker** у C#.



```
<StackLayout Padding="20">
    <Picker x:Name="pick"
            Title="Мови програмування"
            SelectedIndexChanged="OnPickerSelectedIndexChanged">
        <Picker.Items>
            <x:String>C#</x:String>
            <x:String>C/C++</x:String>
            <x:String>JavaScript</x:String>
            <x:String>PHP</x:String>
        </Picker.Items>
    </Picker>

    <Label x:Name="lbl"
           FontSize="Default"
           TextColor="Black"/>
</StackLayout>
```

Рисунок 1.7 – Реалізація елементу **Picker** у XAML

```
public partial class MainPage : ContentPage
{
    Label lbl;
    Picker pick;
    public MainPage()
    {
        lbl = new Label
        {
            Text = "",
            FontSize = Device.GetNamedSize(NamedSize.Default, typeof(Label))
        };
        pick = new Picker
        {
            Title = "Мови програмування"
        };
        pick.Items.Add("C#");
        pick.Items.Add("C/C++");
        pick.Items.Add("JavaScript");
        pick.Items.Add("PHP");
        pick.SelectedIndexChanged += OnPickerSelectedIndexChanged;
        Content = new StackLayout
        {
            Padding = 20,
            Children = { pick, lbl }
        };
    }
    void OnPickerSelectedIndexChanged(object sender, EventArgs e)
    {
        lbl.Text = "Вибрано мову – " + pick.Items[pick.SelectedIndex];
    }
}
```

Рисунок 1.8 – Реалізація елементу **Picker** у C#

1.4 Елемент **Stepper**

Елемент **Stepper** у рамках технології .NET MAUI є одним із стандартних засобів введення числових значень, призначений для забезпечення зручного інтерфейсу користувача при виборі чисел. Його ключова задача – дозволити користувачу вводити числові дані двома способами: або безпосереднім набором через клавіатуру, або ж коригуванням поточного значення за допомогою кнопок збільшення [+] та зменшення [-]. Це робить **Stepper** особливо корисним для сценаріїв, де потрібне точне введення числових параметрів з можливістю швидко змінювати їх крок за кроком.

Основні властивості елемента **Stepper** у MAUI створені для гнучкого налаштування поведінки цього елемента: *Increment* визначає крок, на який відбувається збільшення або зменшення числового значення. За допомогою цього параметра можна встановити лютий темп зміни числа при натисканні кнопок, наприклад, по 0.1, 1, 2 і так далі, що залежить від потреб конкретного застосунку; *Maximum* та *Minimum* задають верхню та нижню межі допустимого діапазону введення. Це гарантує, що користувач не зможе обрати некоректне або неприпустиме значення, забезпечуючи тим самим контроль коректності введених даних прямо на етапі взаємодії з інтерфейсом; *Value* – це властивість, яка зберігає поточне числове значення елемента. Її можна як отримати програмно, так і змінити, що дає можливість реалізовувати логіку адаптації значення за потребою бізнес-логіки застосунку.

Головна подія, яка супроводжує роботу **Stepper** – це *ValueChanged*. Вона спрацьовує у момент змінення властивості *Value*, тобто коли користувач або програмно змінює значення у **Stepper**, ця подія дозволяє виконати додаткові дії, наприклад, оновити відображення значення або виконати обчислення на основі нового числа.

Наступний код на рисунку 1.9 демонструє, як можна інтегрувати Stepper до інтерфейсу користувача за допомогою мови розмітки XAML.

```
<StackLayout>
    <Label x:Name="lbl" Text=" " FontSize="Medium" />
    <Stepper Minimum="0" Maximum="10" Increment="2" ValueChanged="OnStepperValueChanged" />
</StackLayout>
```

Рисунок 1.9 – Реалізація елементу **Stepper** у XAML

У цьому прикладі створюється вертикальне компонування, що містить **Label** для відображення обраного значення і **Stepper** з параметрами, які обмежують значення від 0 до 10 із кроком 2. Подія *ValueChanged* налаштована

на обробник *OnStepperValueChanged*, що викликається при зміні значення. Приклад програмного створення цих елементів на мові C# зображений на рисунку 1.10.

```
lbl = new Label
{
    Text = "",
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    HorizontalOptions = LayoutOptions.Center
};

Stepper stepper = new Stepper
{
    Minimum = 0,
    Maximum = 10,
    Increment = 0.1,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.CenterAndExpand
};

stepper.ValueChanged += OnStepperValueChanged;
this.Content = new StackLayout { Children = { lbl, stepper } };
```

Рисунок 1.10 – Реалізація елементу **Stepper** у C#

Тут створюється **Label** і **Stepper** програмно з подібними параметрами, де **Stepper** має трохи більшу точність кроку – 0.1. Такі зміни дають розробнику гнучкі можливості динамічно будувати інтерфейси. Обробник події *ValueChanged* реалізується у вигляді методу (рис. 1.11).

```
private void OnStepperValueChanged(object sender, ValueChangedEventArgs e)
{
    if (lbl != null)
        lbl.Text = String.Format("Вибрано: {0:F1}", e.NewValue);
}
```

Рисунок 1.11 – Реалізація обробника **Stepper** у C#

Цей метод перевіряє, чи існує **Label**, і у випадку позитивної перевірки оновлює текст у ньому новим значенням з точністю до одного знаку після коми.

1.5 Елемент **Slider**

Елемент **Slider** у технології .NET MAUI призначено для зручного введення числових значень за допомогою інтерактивного повзунка. Такий елемент дозволяє користувачеві задавати значення в межах певного діапазону, переміщаючи повзунок уздовж горизонтальної шкали. **Slider** часто використовується для регулювання гучності, яскравості екрана, рівня масштабування або будь-якого іншого параметра, що має безперервний числовий діапазон. Такий підхід є інтуїтивно зрозумілим і зручним для користувачів, особливо коли потрібно вказати значення, яке не є дискретним (наприклад, гучність, яскравість, об’єм тощо). Компонент **Slider** реалізує інтерфейс *BindableObject*, тому основні його властивості можуть зв’язуватися з моделлю даних у межах концепції MVVM. Це робить його універсальним елементом для сучасних кросплатформних застосунків, де особливо важливо забезпечити узгодженість станів між інтерфейсом користувача та бізнес-логікою.

Для налаштування поведінки та відображення елемента **Slider** використовуються наступні ключові властивості: *Minimum* – визначає найменше значення, яке може бути вибране користувачем. За замовчуванням дорівнює 0. Ця властивість дозволяє обмежити нижню межу діапазону вибору; *Maximum* – визначає найбільше значення, яке може бути вибране користувачем. За замовчуванням дорівнює 1. Встановлення цієї властивості дозволяє задати верхню межу діапазону; *Value* – містить поточне значення, вибране користувачем. Це значення змінюється при пересуванні повзунка і може бути прочитане або змінене програмно.

Головною подією, пов’язаною з елементом **Slider**, є *ValueChanged*. Вона виникає кожного разу, коли користувач змінює положення повзунка, тобто коли змінюється значення властивості *Value*. Обробник цієї події отримує параметр типу *ValueChangedEventArgs*, який містить два поля: *OldValue* (попереднє значення) та *NewValue* (нове значення). Це дозволяє відстежувати зміни в реальному часі і реагувати на них, наприклад, оновлюючи інтерфейс або виконуючи обчислення.

Нижче наведено приклад використання **Slider** у двох варіантах: через XAML (рис. 1.12) та через C# (рис. 1.13).

```xaml
<StackLayout>
    <Label x:Name="lbl" Text=" " FontSize="Large" />
    <Slider Minimum="0" Maximum="50" Value="30"
            ValueChanged="OnSliderValueChanged" />
</StackLayout>
```

Рисунок 1.12 – Реалізація елементу **Slider** у XAML

У цьому прикладі створюється елемент **Slider**, який дозволяє вибрати значення від 0 до 50, а початкове значення встановлено на 30. Подія *ValueChanged* підписана на метод *OnSliderValueChanged*, який буде викликатися при кожній зміні значення.

```csharp
lbl = new Label
{
    Text = "",
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    HorizontalOptions = LayoutOptions.Center
};

Slider slider = new Slider { Minimum = 0, Maximum = 50, Value = 30 };
slider.ValueChanged += OnSliderValueChanged;

this.Content = new StackLayout { Children = { lbl, slider } };
```

Рисунок 1.13 – Реалізація елементу **Slider** у C#

Тут **Slider** створюється програмно, а подія *ValueChanged* підписується на обробник. Код обробника події *ValueChanged* має наступний вигляд (рис. 1.14).

```
void OnSliderValueChanged(object sender, ValueChangedEventArgs e)
{
    if (lbl != null)
        lbl.Text = String.Format("Вибрано: {0:F1}", e.NewValue);
}
```

Рисунок 1.14 – Реалізація обробника події *ValueChanged* елементу **Slider** у C#

У цьому прикладі змінна *lbl* оголошена як глобальна змінна типу **Label**. При кожній зміні значення повзунка відображається оновлене значення у форматі з одним знаком після коми. Це дозволяє користувачеві бачити поточне значення безпосередньо на екрані.

2 РОБОЧЕ ЗАВДАННЯ

Завдання 1. Проаналізувати теоретичні основи розробки кросплатформових додатків за допомогою .NET MAUI, викладені в розділі 1 навчального матеріалу. На основі отриманих знань дослідити функціональні можливості інтегрованого середовища розробки Visual Studio, зокрема інструменти для створення інтерактивних елементів у MAUI-додатках. Практично реалізувати всі наведені у розділі приклади, забезпечивши їх працездатність на принаймні одній цільовій платформі (Windows, Android або iOS).

Завдання 2. Розробити кросплатформовий додаток за допомогою технології .NET MAUI. Головний екран додатка має містити елементи управління **Button**, **Switch**, **Stepper, Picker** та **Slider** із різним форматуванням.

3 КОНТРОЛЬНІ ПИТАННЯ

1. Яке призначення елемента управління **Button** у .NET MAUI?
2. Які основні властивості та події має елемент **Button**?
3. Для чого використовується елемент керування **Switch** у .NET MAUI?
4. Які основні властивості та події характеризують елемент **Switch**?
5. Яке призначення елемента **Picker** у контексті взаємодії з користувачем?
6. Які ключові властивості та події має елемент **Picker**?
7. Для чого застосовується елемент **Stepper** у .NET MAUI?
8. Які основні властивості та події визначають поведінку елемента **Stepper**?
9. Яке функціональне призначення елемента **Slider** у користувацькому інтерфейсі?
10. Які ключові властивості та події має елемент **Slider**?

ПРАКТИЧНА РОБОТА №4

Тема роботи: дослідження можливостей Visual Studio та .NET MAUI: компонування й оформлення інтерфейсу з введенням дати/часу

Мета роботи: освоїти компонування інтерфейсу та оформлення елементів введення дати і часу в середовищі Visual Studio з використанням .NET MAUI для створення зручних кросплатформних додатків.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Елемент інтерфейсу (відомий також як елемент управління, віджет або «контрол») – це базовий компонент графічного інтерфейсу користувача, який має стандартний візуальний вигляд і виконує типові дії. У MAUI (Multi-platform App UI) кожен елемент інтерфейсу розроблений з урахуванням кросплатформності, що дозволяє створювати уніфіковані інтерфейси для Android, iOS, Windows та інших платформ. Елементи інтерфейсу в MAUI можуть бути простими (наприклад, кнопки, мітки, поля введення) або складними контейнерами, які об’єднують інші елементи та надають додаткові можливості для стилізації та організації вмісту.

1.1 Елемент **Frame**

Одним із найпоширеніших контейнерів у MAUI є елемент **Frame**. Він призначений для створення прямокутної межі навколо будь-якого вкладеного вмісту, що дозволяє візуально відокремити блок інформації або інтерактивних елементів. За замовчанням **Frame** автоматично додає внутрішні відступи (*Padding*) у розмірі 20 одиниць, що забезпечує комфортне відображення вмісту всередині рамки. Це робить **Frame** ідеальним для створення карточок, панелей або будь-яких інших блоків, які потребують візуального акцентування. Основні властивості **Frame** є: *CornerRadius* – визначає радіус заокруглення кутів рамки. Значення може бути від 0 (прямокутна рамка) до будь-якого додатного числа, що дозволяє створювати м’які, закруглені кути, характерні

для сучасних інтерфейсів; *HasShadow* – логічна властивість, яка вказує, чи має відображатися тінь навколо рамки. Використання тіні дозволяє візуально відокремити блок від фону, що підвищує читабельність інтерфейсу; *OutlineColor* – задає колір рамки, яка оточує вкладений вміст. Ця властивість дозволяє налаштувати візуальний стиль Frame, щоб він відповідав загальній темі додатку.

Frame можна додавати до сторінки як за допомогою XAML (рис. 1.1), так і програмно через C#. Це забезпечує гнучкість при розробці інтерфейсу та дозволяє використовувати переваги обох підходів.

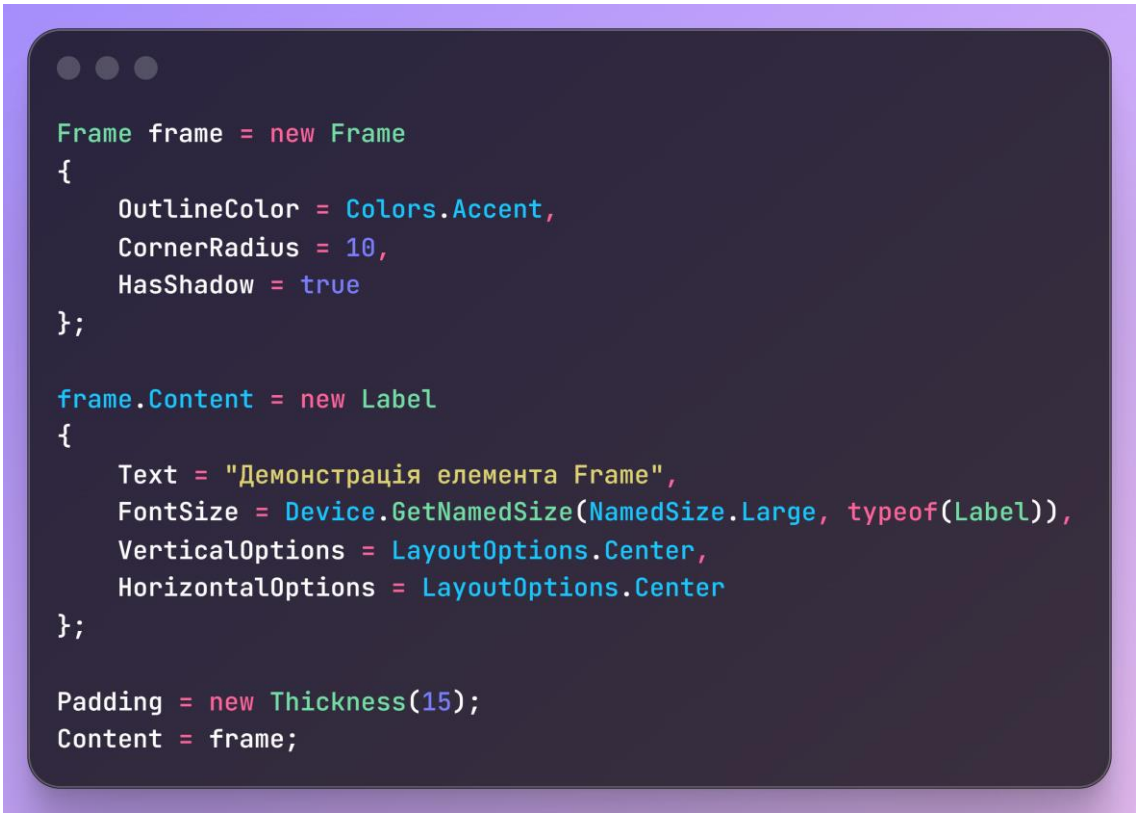
```
<Frame OutlineColor="Accent"
       CornerRadius="10"
       HasShadow="True">
    <Label Text="Демонстрація елемента Frame"
           FontSize="Large"
           VerticalOptions="Center"
           HorizontalOptions="Center"/>
</Frame>
```

Рисунок 1.1 – Приклад створення елементу **Frame** у XAML

У цьому прикладі **Frame** має рамку кольору *Accent*, скруглені кути радіусом 10 одиниць і відображає тінь. Вкладений **Label** розміщується по центру рамки. Аналогічний результат можна отримати, створивши **Frame** програмно в коді C# (рис. 1.2). Цей підхід дозволяє динамічно створювати та налаштовувати **Frame** під час виконання програми, що корисно для складних сценаріїв, коли властивості елементів залежать від даних або дій користувача.

Frame є універсальним інструментом для структурування інтерфейсу. Його можна використовувати для створення карток, повідомлень, форм, блоків інформації та інших компонентів, які потребують візуального

відокремлення. Використання **Frame** дозволяє підвищити якість дизайну додатку, зробити інтерфейс більш зрозумілим і зручним для користувача.



```
Frame frame = new Frame
{
    OutlineColor = Colors.Accent,
    CornerRadius = 10,
    HasShadow = true
};

frame.Content = new Label
{
    Text = "Демонстрація елемента Frame",
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    VerticalOptions = LayoutOptions.Center,
    HorizontalOptions = LayoutOptions.Center
};

Padding = new Thickness(15);
Content = frame;
```

Рисунок 1.2 – Приклад створення елементу **Frame** у C#

1.2 Елемент **BoxView**

Елемент **BoxView** є одним із базових візуальних компонентів у .NET MAUI, призначених для створення простих прямокутних областей, які можуть бути пофарбовані в будь-який колір. Цей елемент часто використовується для візуального розділення частин інтерфейсу, створення декоративних елементів, або для відображення інформації у вигляді кольорових блоків. **BoxView** не підтримує вбудовані текстові або інтерактивні функції, але його можна легко комбінувати з іншими елементами для створення складніших композицій. **BoxView** надає кілька ключових властивостей, які дозволяють гнучко налаштовувати його візуальні характеристики: *Color* – визначає колір заливки прямокутника. Приймає значення типу *Color*, яке може бути задане через

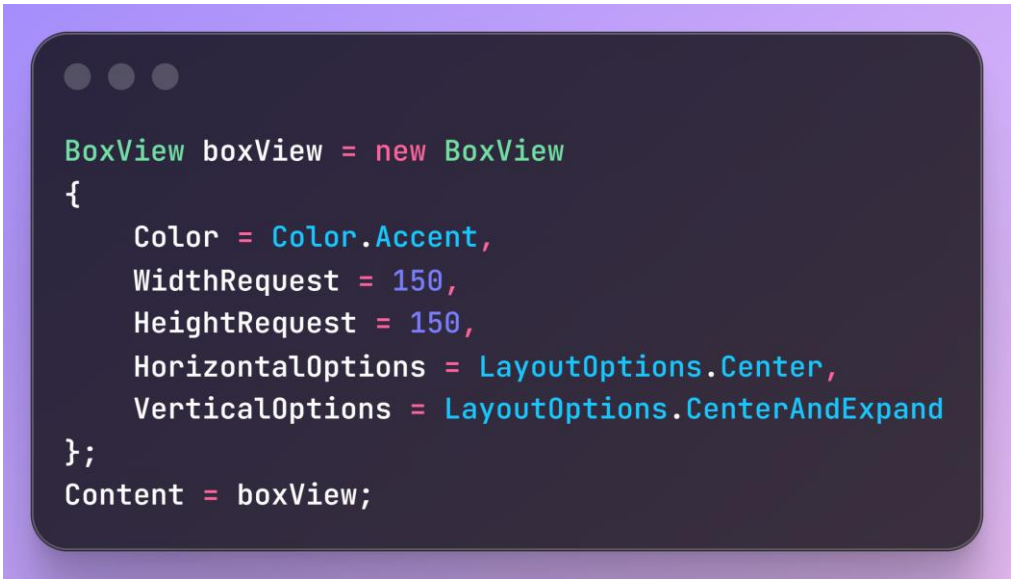
стандартні кольори (наприклад, *Color.Red, Color.Blue*), або через власні RGB-значення; *HeightRequest* – встановлює бажану висоту елемента. За замовчуванням значення становить 40 одиниць (незалежно від платформи, це відповідає умовним одиницям, які автоматично масштабуються під різні роздільності екранів); *WidthRequest* – встановлює бажану ширину елемента. За замовчуванням також дорівнює 40 одиниць; *IsVisible* – логічна властивість, яка визначає, чи буде елемент відображатися на сторінці. Якщо значення false, BoxView не буде показаний, але залишиться у розмітці. *Opacity* – встановлює рівень непрозорості елемента (від 0.0 – повністю прозорий, до 1.0 – повністю непрозорий). Це дозволяє створювати ефекти напівпрозорості.

BoxView можна додавати до сторінки як через XAML (рис. 1.3), так і через C# (рис. 1.4). Це забезпечує гнучкість при розробці інтерфейсу, дозволяючи використовувати найбільш зручний для конкретного випадку підхід.

Рисунок 1.3 – Приклад створення елементу **BoxView** у XAML

У цьому прикладі створюється **BoxView**, який заповнюється кольором «Accent» (системний акцентний колір), і розміщується по центру сторінки за вертикаллю та горизонталлю.

Тут **BoxView** створюється програмно, задаються його розміри та вирівнювання, після чого він призначається як основний вміст сторінки.



```
BoxView boxView = new BoxView
{
    Color = Color.Accent,
    WidthRequest = 150,
    HeightRequest = 150,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.CenterAndExpand
};
Content = boxView;
```

Рисунок 1.4 – Приклад створення елементу **BoxView** у C#

BoxView часто використовується разом з іншими елементами для створення складних композицій (рис. 1.5). Наприклад, можна розмістити кілька **BoxView** у **StackLayout**, щоб відобразити кольорові блоки разом із текстовими мітками.

```
<StackLayout>
    <StackLayout Orientation="Horizontal">
        <BoxView Color="Red" WidthRequest="50" HeightRequest="50" />
        <Label Text="Red" FontSize="25" />
    </StackLayout>
    <StackLayout Orientation="Horizontal">
        <BoxView Color="Yellow" WidthRequest="50" HeightRequest="50" />
        <Label Text="Yellow" FontSize="25" />
    </StackLayout>
    <StackLayout Orientation="Horizontal">
        <BoxView Color="Green" WidthRequest="50" HeightRequest="50" />
        <Label Text="Green" FontSize="25" />
    </StackLayout>
</StackLayout>
```

Рисунок 1.5 – Приклад розміщення кількох елементів **BoxView** у XAML

BoxView є важливим елементом для розробки кросплатформних додатків у MAUI, оскільки дозволяє швидко створювати візуальні ефекти та

інтерфейсні компоненти, які виглядають однаково на всіх підтримуваних платформах.

1.3 Елемент **DatePicker**

Елемент **DatePicker** є одним із ключових компонентів користувацького інтерфейсу .NET MAUI, призначеним для забезпечення інтуїтивного та зручного способу вибору дат у кросплатформних додатках. Цей елемент управління автоматично викликає нативний системний інструмент вибору дати на кожній платформі (iOS, Android, Windows, macOS), що забезпечує користувачам знайомий та природний досвід роботи відповідно до звичок їхньої операційної системи. **DatePicker** забезпечує введення валідної дати без ручного парсингу рядків, обмеженням діапазонів і локалізованим форматуванням відображення за стандартами .NET, що знижує кількість помилок введення та спрощує валідацію форм. Вибір дати відбувається через нативний діалог/випадний інтерфейс платформи, після чого контроль оновлює властивість *Date* і піднімає подію *DateSelected* із даними про старе та нове значення. Використання **DatePicker** є особливо важливим у сценаріях, де необхідно отримати від користувача точну дату, наприклад, при бронюванні квитків, заповненні форм реєстрації, плануванні подій чи введенні дати народження. Елемент **DatePicker** у .NET MAUI надає розробникам широкий набір властивостей для налаштування його поведінки та зовнішнього вигляду. Основні функціональні властивості включають можливості обмеження діапазону доступних дат та управління відображенням обраної дати: *MinimumDate* – мінімально дозволена дата; за замовчуванням – перший день 1900 року, тип *DateTime*; *MaximumDate* – максимально дозволена дата; за замовчуванням – останній день 2100 року, тип *DateTime*; *Date* – обрана дата, за замовчуванням – *DateTime.Today*, оновлюється при виборі користувачем; *Format* – рядок стандартного або користувацького .NET-формату, за

замовчуванням "D" (довгий формат дати); *TextColor* – колір текстового представлення дати, застосовується до відображуваного рядка.

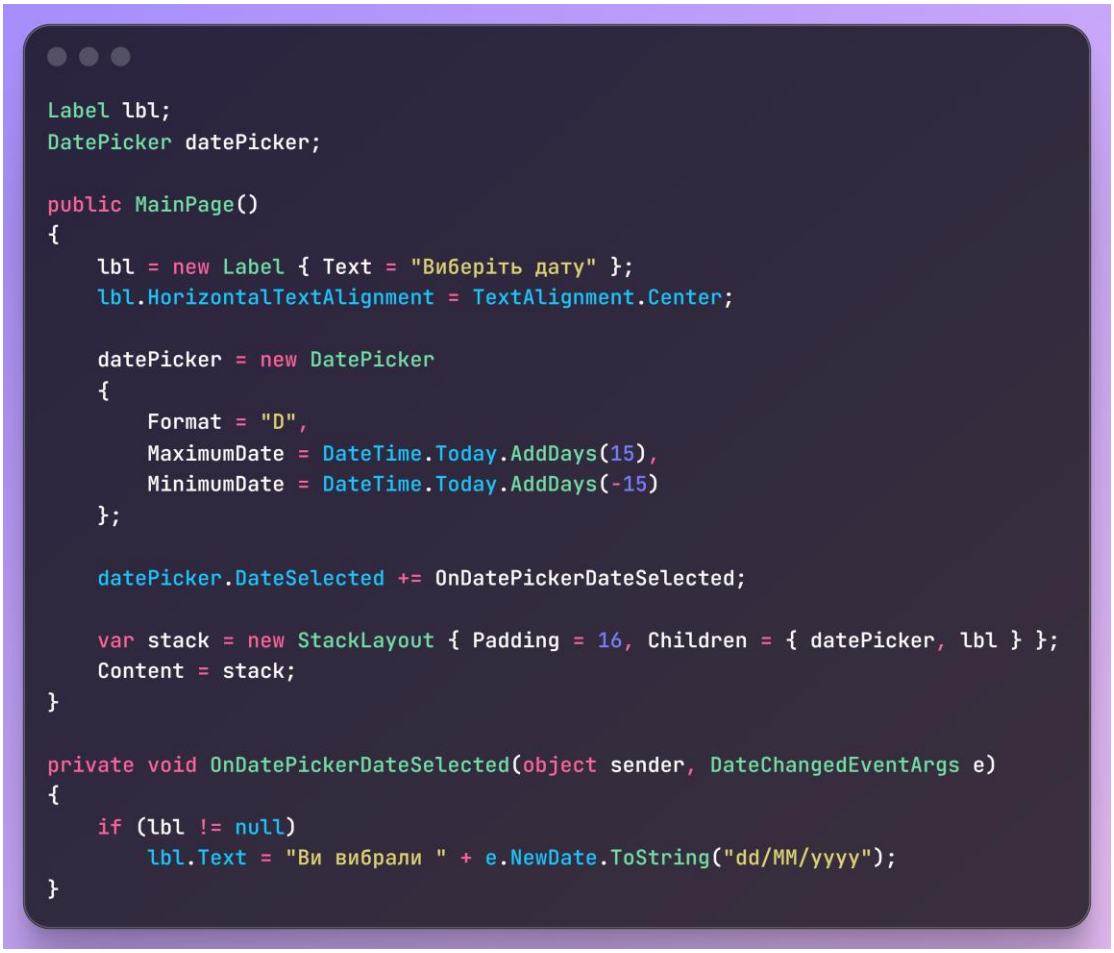
Додатково: підтримуються типові шрифтові властивості *FontAttributes*, *FontFamily*, *FontSize ma CharacterSpacing* для налаштування вигляду тексту.

Основною подією є *DateSelected* – стандартний *EventHandler<DateChangedEventArgs>* із властивостями *OldDate* та *NewDate*, що дозволяє реагувати на зміну значення та оновлювати інтерфейс або бізнес-логіку форми в момент підтвердження вибору. Слід враховувати UX-нюанси: якщо користувач закриває **DataPicker** без зміни дати або повторно обирає поточну дату, подія може не підніматися залежно від платформи та версії, тому критично важливі сценарії краще страхувати прив'язками і валідацією стану. Приклад створення елементу **DatePicker** наведений на рис. 1.6.

```
<StackLayout Padding="16">
  <DatePicker
      x:Name="RangeStartPicker"
      Format="D"
      MinimumDate="1900-01-01"
      MaximumDate="2100-12-31"
      DateSelected="OnRangeStartSelected" />
  <DatePicker
      x:Name="RangeEndPicker"
      Format="D"
      MinimumDate="1900-01-01"
      MaximumDate="2100-12-31"
      DateSelected="OnRangeEndSelected" />
  <Label x:Name="lbl" Text="Виберіть дату" FontSize="Medium" />
</StackLayout>
```

Рисунок 1.6 – Створення елементу **DatePicker** у XAML

Цей приклад задає чіткі глобальні межі, використовує довгий формат «D» і під'єднує обробники для синхронізації меж між періодами. Реалізація цього ж прикладу на мові програмування C# показана рисунку 1.7.



```
Label lbl;
DatePicker datePicker;

public MainPage()
{
    lbl = new Label { Text = "Виберіть дату" };
    lbl.HorizontalTextAlignment = TextAlignment.Center;

    datePicker = new DatePicker
    {
        Format = "D",
        MaximumDate = DateTime.Today.AddDays(15),
        MinimumDate = DateTime.Today.AddDays(-15)
    };

    datePicker.DateSelected += OnDatePickerDateSelected;

    var stack = new StackLayout { Padding = 16, Children = { datePicker, lbl } };
    Content = stack;
}

private void OnDatePickerDateSelected(object sender, DateChangedEventArgs e)
{
    if (lbl != null)
        lbl.Text = "Ви вибрали " + e.NewDate.ToString("dd/MM/yyyy");
}
```

Рисунок 1.7 – Створення елементу **DatePicker** у C#

У коді задаються динамічні межі відносно сьогоднішньої дати, використовується подія *DateSelected* і форматування для наочного відображення вибраного значення в **Label**.

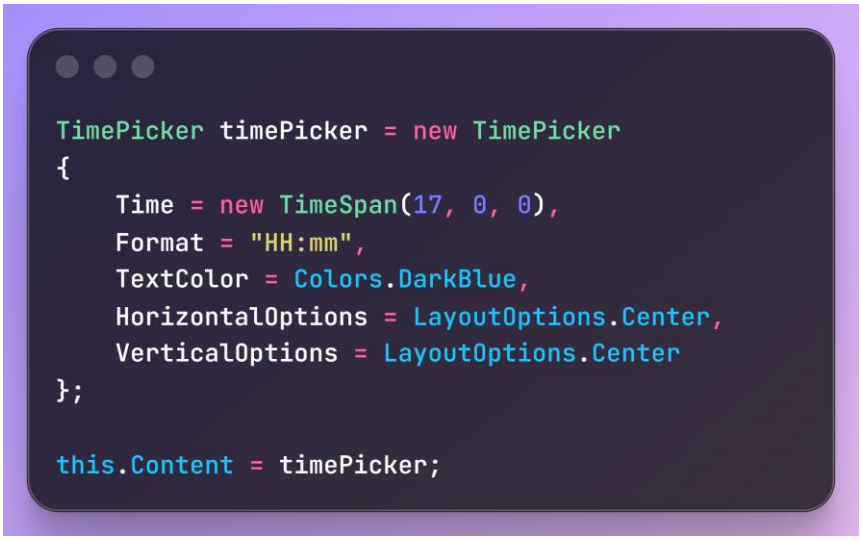
1.4 Елемент **TimePicker**

Елемент **TimePicker** у технології .NET MAUI використовується для вибору часу користувачем у зручному візуальному форматі. Цей елемент забезпечує єдину поведінку на різних платформах – Android, iOS, macOS і Windows, автоматично адаптуючи свій зовнішній вигляд під стиль та системні компоненти кожної ОС. Таким чином, розробник отримує узгоджений інтерфейс елемента вибору часу на будь-якому пристрої, не створюючи окремих компонентів під кожну платформу.

TimePicker відображає часовий інтерактивний елемент (як правило, це колесо, список або системне діалогове вікно), що дозволяє користувачеві обрати години та хвилини. Основна відмінність від елемента **DatePicker** полягає в тому, що **TimePicker** не оперує датами, а лише моментом у межах доби. Основні властивості елемента **TimePicker**: *Format* – властивість, яка визначає текстовий формат відображення часу. Наприклад, шаблон «HH:mm» покаже час у 24-годинному форматі, а «hh:mm tt» – у 12-годинному з індикатором AM/PM. Формат задається у вигляді рядка за стандартними правилами форматування дати й часу .NET; *TextColor* – визначає колір тексту, у якому буде відображено вибраний час. Приймає значення типу *Color*; *Time* – задає або повертає вибраний час у вигляді об’єкта типу *TimeSpan*. Ця властивість є основною для зберігання та оброблення значення часу.

Найменша одиниця вимірювання в *TimeSpan* – це «тік» («tick»). Один тік дорівнює 100 наносекундам, тобто 10^{-7} секунди. Таким чином, один день у системі TimeSpan дорівнює 864000000000 тіків. Діапазон можливих значень змінюється від *TimeSpan.MinValue* до *TimeSpan.MaxValue*, що забезпечує надзвичайно високу точність у вимірюванні часу. Важливо розуміти, що структура *TimeSpan* не призначена для зберігання абсолютних моментів у часі (наприклад, дати й години події). Якщо вам потрібно врахувати дату разом із часом, варто використовувати *DateTime* або *DateTimeOffset*, які містять як дату, так і часову зону.

Відображення елемента **TimePicker** показано на рисунку 1.8. У наведеному прикладі створюється елемент **TimePicker**, який відображає початковий час 17:00 у 24-годинному форматі, центрований по горизонталі та вертикалі на сторінці. Користувач може взаємодіяти з компонентом, змінюючи значення, після чого властивість **Time** автоматично оновиться, що можна відстежити у коді або через прив’язування даних.



```
TimePicker timePicker = new TimePicker
{
    Time = new TimeSpan(17, 0, 0),
    Format = "HH:mm",
    TextColor = Colors.DarkBlue,
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center
};

this.Content = timePicker;
```

Рисунок 1.8 – Створення елементу **TimePicker** у C#

2 РОБОЧЕ ЗАВДАННЯ

Завдання 1. Проаналізувати теоретичні основи розробки кросплатформових додатків за допомогою .NET MAUI, викладені в розділі 1 навчального матеріалу. На основі отриманих знань дослідити функціональні можливості інтегрованого середовища розробки Visual Studio, зокрема інструменти для створення елементів компонування й оформлення інтерфейсу з введенням дати/часу у MAUI-додатках. Практично реалізувати всі наведені у розділі приклади, забезпечивши їх працездатність на принаймні одній цільовій платформі (Windows, Android або iOS).

Завдання 2. Розробити кросплатформовий додаток за допомогою технології .NET MAUI. Головний екран додатка має містити елементи управління **Frame, BoxView, DatePicker** та **TimePicker** із різним форматуванням.

3 КОНТРОЛЬНІ ПИТАННЯ

1. Яке призначення елемента **Frame** у застосунках .NET MAUI?
2. Які основні властивості та характеристики притаманні елементу **Frame**?
3. Яке функціональне призначення елемента **BoxView**?

4. Які ключові властивості визначають поведінку та зовнішній вигляд елемента **BoxView**?

5. Для чого використовується елемент **DatePicker** у .NET MAUI?

6. Які основні властивості та можливості підтримує елемент DatePicker?

7. Яка роль елемента **TimePicker** у розробці користувацьких інтерфейсів MAUI?

8. Які властивості визначають поведінку та параметри елемента **TimePicker**?

ПРАКТИЧНА РОБОТА №5

Тема роботи: дослідження можливостей Visual Studio та .NET MAUI: робота з зображеннями.

Мета роботи: навчитися створювати і використовувати зображення в середовищі Visual Studio з використанням .NET MAUI.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Елемент **Image**

Елемент **Image** у .NET MAUI використовується для відображення графічних зображень у мобільних, десктопних та вебзастосунках. Він є одним з базових елементів інтерфейсу користувача та входить до складу простору імен *Microsoft.Maui.Controls*. Основне його призначення – завантаження та виведення на екран зображень із різних джерел, таких як локальні ресурси, вбудовані файли, віддалені URL або потоки даних (stream). У .NET MAUI **Image** реалізовано як гнучкий і кросплатформний компонент: він автоматично адаптується до особливостей кожної платформи – Android, iOS, Windows чи macOS. Це означає, що однаковий XAML-код для **Image** буде коректно відображатися на всіх цільових платформах без необхідності створювати окремі реалізації.

Основні властивості елемента **Image**: *Aspect* – визначає, як зображення має масштабуватися та відображатися всередині відведеної області. У .NET MAUI ця властивість приймає значення із перерахування *Aspect*, яке описує режим вирівнювання та пропорційність відображення зображення. *AspectFit* – зображення масштабується так, щоб повністю вміститися у вказану область, при цьому зберігаються його початкові пропорції (аспектне відношення). Може з’явитися порожній простір навколо зображення, якщо співвідношення сторін елемента та зображення різні. Це стандартне значення властивості. *Fill* – зображення розтягується по ширині та висоті, щоб повністю заповнити

відведену область. Аспектне відношення не зберігається, тому зображення може виглядати спотвореним, якщо пропорції не збігаються. *AspectFill* – зображення масштабується з урахуванням збереження пропорцій, але обрізається таким чином, щоб повністю заповнити область. Частина зображення може бути невидимою за межами контейнера. Вибір режиму залежить від того, що важливіше в конкретному випадку – збереження пропорцій зображення чи повне заповнення простору користувацького інтерфейсу. *Source* – задає джерело зображення. У .NET MAUI вона підтримує кілька типів джерел, об'єднаних класом ImageSource. Це дозволяє гнучко вибирати спосіб завантаження графіки в застосунок. *FileImageSource* – використовується для зображень, що зберігаються локально у файловій системі або в ресурсах застосунку; *UriImageSource* – завантажує зображення з мережі за вказаним URL; *StreamImageSource* – дозволяє передати зображення зі струму байтів (наприклад, із бази даних або пам'яті); *EmbeddedResourceImageSource* – використовується, якщо зображення вбудоване безпосередньо в збірку. *IsLoading* – має булевий тип і вказує, чи відбувається наразі завантаження зображення. Це зручно для організації візуального зворотного зв'язку (наприклад, показу індикатора завантаження). Особливо актуально при завантаженні ресурсів із мережі через UriImageSource.IsOpaque – визначає, чи має зображення прозорі області. Якщо встановлено true, елемент вважається непрозорим, що може вплинути на оптимізацію рендерингу. Для зображень із прозорими фонами (PNG, WebP) зазвичай використовується значення false.

Елемент **Image** у MAUI підтримує роботу з кешуванням при завантаженні з мережевих джерел, асинхронне підключення, а також інтеграцію з **Layout**-елементами для адаптивного розташування контенту. Крім того, він добре поєднується з такими класами, як *ImageButton*, який розширює функціонал стандартного **Image**, додаючи обробку подій натискання.

1.2 Робота з локальними зображеннями

У .NET MAUI робота з локальними зображеннями дещо спрощена завдяки централізованому розміщенню ресурсів у папці *Resources\Images* проекту. Для додавання локальних зображень до проекту необхідно просто помістити файли до цієї папки. При цьому система автоматично встановлює для них тип збірки MauiImage, що дозволяє .NET MAUI правильно обробляти і підбирати зображення під різні платформи та роздільні здатності пристроїв. Варто пам’ятати, що назви файлів мають відповідати правилам операційної системи Android: повинні бути лише маленькими літерами, починатися і закінчуватися літерою, і містити лише цифри, літери або підкреслення.

Коли проект збирається, .NET MAUI на основі ресурсу у Resources\Images створює усі необхідні версії зображення для різних платформ, підтримуючи автоматичний вибір потрібного варіанту в залежності від роздільної здатності екрана пристрою. Це дозволяє уникнути дублювання файлів для різних платформ і спрощує керування ресурсами у проекті.

Щоб відобразити локальне зображення в UI, можна скористатися двома основними способами:

- через XAML, де достатньо задати тег **Image** із вказівкою імені файлу у властивості *Source*, наприклад*: <Image Source="auto.jpg" />*
- через C#, створивши об’єкт **Image** і задавши йому *Source* (рис. 1.1).

```
Image image = new Image { Source = "auto.jpg" };
this.Content = image;
```

Рисунок 1.1 – Створення зображення на мові C#

Існують і платформоспецифічні нюанси у роботі з локальними зображеннями. Для Android важливо розміщувати файли відповідно до їх щільності у підпапках *drawable-hdpi, drawable-xhdpi, drawable-xxhdpi* тощо, що відповідає різним dpi пристроїв. У iOS система використовує суфікси @2x, @3x у назвах файлів для позначення вищої роздільної здатності, наприклад primer.jpg, primer@2x.jpg, primer@3x.jpg. UWP просто допускає додавання зображень у кореневий каталог проекту з відповідною дією збірки *Content*.

Щоб враховувати відмінності шляху до ресурсів та особливості платформи, .NET MAUI підтримує метод *OnPlatform* (хоча він дещо застарілий), який дозволяє заздалегідь вказати різні шляхи або поведінку для Android, iOS і Windows. Приклад у C# для встановлення джерела зображення з урахуванням платформи може виглядати так (рис. 1.2).

Рисунок 1.2 – Створення зображення на мові C# в залежності від платформи

Загалом, сучасний підхід у .NET MAUI полягає у використанні централізованої папки *Resources\Images*, де зображення для всіх платформ зберігаються, а платформа сама дбає про масштабування, підбір оптимального файла і правильне включення ресурсів до збірки проекту. Це покращує кросплатформенність, зручність підтримки і дозволяє ефективно працювати з локальними зображеннями у додатках. Також при створенні інтерфейсів зображення варто враховувати різні розміри екранів і dpi пристроїв, щоб забезпечити коректний вигляд елементів UI на всіх платформах. Для цього

слід підготувати кілька версій кожного растрового файлу на різних роздільних здатностях, щоб користувачі отримували найкращу якість без зайвого навантаження на пам’ять і продуктивність пристрою.

1.3 Робота з вбудованими зображеннями

Вбудовані зображення у .NET MAUI представляють альтернативний підхід до організації графічних ресурсів додатку порівняно зі стандартним механізмом *MauiImage*. На відміну від локальних зображень, які розміщуються у папці *Resources\Images* і автоматично обробляються системою збірки, вбудовані зображення додаються безпосередньо у збірку як ресурси типу *Embedded Resource*. Це означає, що файл зображення інтегрується безпосередньо в DLL-файл проекту під час компіляції, що робить його невід'ємною частиною збірки. Ключова перевага використання вбудованих зображень полягає у спрощенні структури проекту – достатньо додати файл зображення лише один раз у головний (Portable) проект, без необхідності створювати окремі копії для кожної платформи. Це особливо зручно для зображень, які залишаються ідентичними незалежно від цільової платформи, таких як логотипи, іконки або декоративні елементи інтерфейсу. Додатково, вбудовані ресурси захищені від випадкового видалення або модифікації користувачем після розгортання додатку, оскільки вони фізично не існують як окремі файли у файловій системі пристрою.

Для підготовки вбудованого зображення необхідно створити у головному проекті каталог (наприклад, **Images**) і розмістити у ньому потрібний файл зображення. Після додавання файлу слід відкрити його властивості через контекстне меню (правий клік миші на файлі) та у полі «Build Action» (дія при збірці) встановити значення «EmbeddedResource» (вбудований ресурс). Цей крок критично важливий, оскільки він вказує компілятору обробити файл як вбудований ресурс, а не як звичайний файл

проекту. Якщо цей параметр не встановлений правильно, зображення не буде доступне через потік ресурсів під час виконання програми.

Для програмного доступу до вбудованого зображення у коді C# застосовується статичний метод ImageSource.FromResource(), який приймає як параметр рядок із повним кваліфікованим ідентифікатором ресурсу. Формат цього ідентифікатора має наступну структуру: *<ім'я_проекту>.<ім'я_папки>.<ім'я_файлу>*, де всі компоненти розділяються крапками. Наприклад, якщо проект має назву **FirstApp**, зображення розміщене у папці *Images*, а файл називається *img7.jpg*, повний ідентифікатор буде виглядати як *FirstApp.Images.img7.jpg*. Додавання вбудованого зображення до додатку MAUI показана на рисунку 1.3.

```
public MainPage()
{
    Image image = new Image();
    image.Source = ImageSource.FromResource("FirstApp.Images.img7.jpg");
    this.Content = image;
}
```

Рисунок 1.3 – Створення вбудованого зображення на мові C#

Метод FromResource() повертає об'єкт типу *StreamImageSource*, який внутрішньо зчитує дані зображення з вбудованого потоку ресурсів збірки. Важливо відзначити, що назва проекту, папки та файлу є регістрочутливими, і будь-яка неточність у цьому ідентифікаторі призведе до виключення під час виконання програми через неможливість знайти вказаний ресурс. У .NET MAUI, на відміну від Xamarin.Forms, стандартна розмітка XAML не має вбудованої можливості безпосередньо працювати з ідентифікаторами вбудованих ресурсів. Для вирішення цієї проблеми розроблено офіційний компонент .NET MAUI Community Toolkit, який надає конвертер *ImageResourceConverter* для автоматичного перетворення рядкових

ідентифікаторів ресурсів у відповідні об'єкти *ImageSource*. Для використання *ImageResourceConverter* необхідно спочатку додати простір імен *toolkit* до кореневого елемента XAML-сторінки (рис. 1.4).

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
             x:Class="FirstApp.MainPage">
```

Рисунок 1.4 – Підключення простору імен toolkit у XAML

Після цього конвертер оголошується як ресурс сторінки або додатку (рис. 1.5). Використання конвертера у розмітці відбувається через механізм прив'язки даних із явним зазначенням конвертера.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ImageResourceConverter x:Key="ImageResourceConverter" />
    </ResourceDictionary>
</ContentPage.Resources>
```

Рисунок 1.5 – Оголошення toolkit у якості ресурсу сторінки

1.4 Робота з зображеннями з мережі

У .NET MAUI робота із зображеннями з мережі є важливою частиною розробки сучасних кросплатформних додатків, оскільки багато застосунків потребують динамічного завантаження контенту з віддалених серверів. На відміну від локальних зображень, які зберігаються безпосередньо в проєкті, мережеві зображення завантажуються під час виконання програми, що

дозволяє оновлювати візуальний контент без перевипуску додатку. Для завантаження зображень з мережі в .NET MAUI використовується клас *UriImageSource*, який спеціалізується на роботі з віддаленими ресурсами через протоколи HTTP та HTTPS. Цей клас автоматично обробляє асинхронне завантаження, кешування та відображення зображень, що значно спрощує роботу розробника. Базове використання *UriImageSource* показано на рисунку 1.6.

Рисунок 1.6 – Використання класу *UrlImageSource* для завантаження зображень з мережі

Цей код створює новий елемент **Image** і призначає йому джерело зображення у вигляді URI-адреси. Властивість *CachingEnabled* визначає, чи буде завантажене зображення збережено в локальному кеші для подальшого використання.

Щодо кешування зображень: кешування є критично важливою функцією при роботі з мережевими зображеннями, оскільки воно суттєво впливає на продуктивність додатку та витрати трафіку користувача. Коли кешування увімкнено, .NET MAUI зберігає завантажене зображення у тимчасовій пам'яті пристрою, і при повторному зверненні до того самого URI зображення буде взято з кешу замість повторного завантаження з мережі. Це зменшує навантаження на мережу, прискорює відображення контенту та покращує

загальний користувацький досвід, особливо при повільному інтернет-з'єднанні. Клас *UriImageSource* надає три основні властивості для керування поведінкою завантаження та кешування. Властивість Uri типу *System.Uri* вказує повну адресу зображення в мережі, включаючи протокол (зазвичай HTTPS для безпечного з'єднання). Властивість *CachingEnabled* булевого типу визначає, чи буде активовано механізм кешування для даного зображення. Встановлення значення true увімкне збереження завантаженого зображення, тоді як false змусить додаток завантажувати зображення щоразу заново. Властивість *CacheValidity* типу *TimeSpan* дозволяє точно контролювати тривалість зберігання закешованого зображення.

За замовчуванням, якщо не вказано інше, .NET MAUI зберігає кешовані зображення протягом 24 годин. Цей період є компромісом між актуальністю контенту та ефективністю використання ресурсів. Однак у багатьох реальних сценаріях може знадобитися інший період кешування. Наприклад, статичні логотипи або банери можна кешувати на тривалий час, тоді як новинні фотографії повинні оновлюватися частіше. Для налаштування користувацького періоду кешування використовується підхід, що зображений на рисунку 1.7.

```
Image image = new Image();
image.Source = new UriImageSource
{
    CachingEnabled = true,
    CacheValidity = new TimeSpan(2, 0, 0, 0), // кешування діє 2 дні
    Uri = new Uri("https://example.com/image.jpg")
};
```

Рисунок 1.7 – Налаштування користувацького періоду кешування

Конструктор *TimeSpan* приймає чотири параметри у форматі (дні, години, хвилини, секунди). У наведеному прикладі зображення буде

зберігатися в кеші протягом двох днів. Після закінчення цього терміну при наступному зверненні до зображення MAUI автоматично завантажить свіжу версію з мережі та оновить кеш. Такий підхід особливо корисний для зображень, які оновлюються з певною періодичністю, але не потребують миттєвого оновлення при кожному відкритті екрану.

При роботі з мережевими зображеннями важливо враховувати декілька аспектів. По-перше, завжди використовуйте протокол HTTPS замість HTTP для забезпечення безпеки даних користувачів, оскільки деякі платформи можуть блокувати незахищені з'єднання за замовчуванням. По-друге, обов'язково обробляйте ситуації, коли зображення не вдається завантажити через відсутність інтернет-з'єднання або помилки сервера, надаючи альтернативне локальне зображення через властивість *Image.ErrorPlaceholder* (доступна з .NET 8). По-третє, для оптимізації продуктивності уникайте завантаження зображень надмірно високої роздільності, якщо вони відображатимуться в малих розмірах на екрані.

Також варто пам'ятати про обмеження кешу на різних платформах. Операційні системи можуть автоматично очищати кеш додатку при недостатньому вільному місці, тому критично важливі зображення краще зберігати локально в проєкті. Для складних сценаріїв, які потребують більш гнучкого керування кешуванням або обробки зображень перед відображенням, розгляньте використання сторонніх бібліотек, таких як *FFImageLoading.Maui* або *CommunityToolkit.Maui*.

2 РОБОЧЕ ЗАВДАННЯ

Завдання 1. Проаналізувати теоретичні основи розробки кросплатформових додатків за допомогою .NET MAUI, викладені в розділі 1 навчального матеріалу. На основі отриманих знань дослідити функціональні можливості інтегрованого середовища розробки Visual Studio, зокрема інструменти для створення зображень у MAUI-додатках. Практично

реалізувати всі наведені у розділі приклади, забезпечивши їх працездатність на принаймні одній цільовій платформі (Windows, Android або iOS).

Завдання 2. Розробити кросплатформовий додаток за допомогою технології .NET MAUI, який відображає фотографії автомобілів із колекції локальних зображень (мінімальна кількість – **три**). Перегляд нової фотографії має здійснюватися під час натискання кнопки «**Показати**». Інтерфейс користувача необхідно створити з використанням XAML, а логіку роботи програми реалізувати мовою C#.

3 КОНТРОЛЬНІ ПИТАННЯ

1. Яке призначення елемента **Image** у застосунках .NET MAUI?
2. Які основні властивості елемента **Image** та яке їх функціональне призначення?
3. Яким чином у .NET MAUI реалізується робота з локальними зображеннями?
4. Яку роль виконує метод Device.OnPlatform() під час платформа-залежного відображення зображень?
5. Як здійснюється доступ і відображення вбудованих (embedded) зображень у застосунках .NET MAUI?
6. У яких випадках використовується метод ImageSource.FromResource() та як він функціонує?
7. Як реалізується завантаження та відображення зображень із мережевих ресурсів?
8. Яке призначення об’єктів *UriImageSource* у .NET MAUI та які основні властивості вони мають?

ПРАКТИЧНА РОБОТА №6

Тема роботи: дослідження можливостей Visual Studio та .NET MAUI для роботи з таймерами.

Мета роботи: отримати практичні навички для створення і використання таймерів в середовищі Visual Studio з використанням .NET MAUI.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Таймери є невід'ємним інструментом у розробці сучасних мобільних застосунків, оскільки багато завдань вимагають виконання певних операцій із заданою періодичністю або відстеження поточного часу. Особливо актуальним це стає при створенні застосунків, що працюють у реальному масштабі часу, де своєчасна реакція на зміну стану системи є критично важливою. Таймер являє собою програмний механізм, який забезпечує багаторазове виконання заданого фрагмента коду через певні часові інтервали. На відміну від простої затримки виконання, таймер створює повторюваний цикл, де код виконується автоматично без необхідності ручного запуску кожної ітерації. Це дозволяє розробникам реалізувати асинхронну логіку, яка не блокує основний потік застосунку та не заморожує користувацький інтерфейс.

Принцип роботи таймера полягає у встановленні інтервалу та прив'язці до нього певного обробника подій або callback-функції. Після запуску таймер відраховує заданий часовий проміжок і автоматично викликає вказаний код. Цей процес повторюється циклічно до тих пір, поки таймер не буде зупинено або видалено з пам'яті. Важливо розуміти, що точність спрацювання таймера залежить від завантаженості системи та може варіюватися в межах кількох мілісекунд, що слід враховувати при проектуванні застосунків.

У Xamarin.Forms кожна платформа (Android, iOS, Windows) надавала власну нативну реалізацію таймерів із специфічними особливостями та обмеженнями. Фреймворк Xamarin.Forms створював абстракцію над цими

платформо-специфічними таймерами, забезпечуючи єдиний API для роботи з часовими інтервалами незалежно від цільової платформи. Це спрощувало розробку кросплатформних застосунків, але іноді вимагало додаткових налаштувань для досягнення оптимальної продуктивності на різних пристроях. .NET MAUI, як еволюція Xamarin.Forms, зберігає концепцію кросплатформних таймерів, але впроваджує сучасніші підходи до їх реалізації. Фреймворк використовує уніфіковану систему диспетчеризації подій, яка забезпечує більш передбачувану поведінку таймерів на всіх підтримуваних платформах. При цьому .NET MAUI надає розробникам гнучкість у виборі способу роботи з таймерами, пропонуючи як класичні підходи через *Device.StartTimer* (тепер *Dispatcher.StartTimer*), так і сучасні асинхронні патерни з використанням Task-based API.

У .NET MAUI існує декілька способів застосування таймерів, кожен з яких має свої переваги та оптимальні сценарії використання. Найпростішим є використання вбудованого методу *Dispatcher.StartTimer*, який ідеально підходить для простих періодичних завдань і автоматично виконується в UI-потоці. Для більш складних сценаріїв розробники можуть використовувати *System.Threading.Timer*, який надає точніший контроль над часовими інтервалами та виконується в фоновому потоці. Альтернативним підходом є застосування System.Timers.Timer.

1.1 Використання Dispatcher.StartTimer()

Спосіб реалізації таймерів у .NET MAUI базується на використанні методу-розширення Dispatcher.StartTimer(), який визначений у просторі імен *Microsoft.Maui.Dispatching*. Цей метод дозволяє запустити повторюваний таймер у контексті диспетчера потоків, що автоматично забезпечує виконання callback-функції в правильному потоці для взаємодії з UI-елементами. Метод має таке визначення: *Dispatcher.StartTimer(TimeSpan interval, Func<bool> callback)*. Перший параметр *interval* типу **TimeSpan** визначає часовий інтервал

між викликами таймера. Другий параметр *callback* являє собою функцію зворотного виклику, яка виконуватиметься після кожного спливання заданого інтервалу часу. Особливістю цієї функції є те, що вона повинна повертати булеве значення, яке керує подальшою роботою таймера: true – таймер продовжує працювати і буде викликати callback-функцію після кожного наступного інтервалу; false – таймер зупиняється і більше не виконує зворотні виклики. Така логіка дозволяє гнучко керувати життєвим циклом таймера безпосередньо з коду callback-функції, наприклад, зупиняючи таймер при досягненні певної умови або після виконання заданої кількості ітерацій.

Розглянемо практичний приклад реалізації таймера в .NET MAUI, який демонструє оновлення поточного часу на кнопці з інтервалом в одну секунду (рис. 1.1). Цей додаток дозволяє користувачу запускати та зупиняти таймер за допомогою кнопки **Button**, на якій відображається або поточний час, або повідомлення про стан таймера. Клас *MainPage* успадковується від *ContentPage*, що є базовим контейнером MAUI для одиничної сторінки програми. Ключова особливість – модифікатор partial, який дозволяє розділити реалізацію класу на кілька файлів. Це особливо корисно при роботі з генерованим кодом або у складних проектах, де відділення логіки від XAML-представлення покращує читаність та супроводжуваність коду. Приватні поля класу відіграють роль моделі стану програми. Поле timerbtn зберігає посилання на елемент керування **Button**, що дозволяє звертатися до нього з різних методів класу. Логічна змінна *start* служить прапорцем, який контролює, активний таймер чи ні. Цей простий механізм стану демонструє важливість явного управління життєвим циклом компонентів у MAUI.

```
public partial class MainPage : ContentPage
{
    Button timerbtn;
    bool start = false;

    public MainPage()
    {
        timerbtn = new Button
        {
            Text = "Таймер вимкнено",
            VerticalOptions = LayoutOptions.Center,
            HorizontalOptions = LayoutOptions.Center,
            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Button))
        };
        timerbtn.Clicked += TimerButton_Clicked;
        Content = timerbtn;
    }

    private bool OnTimerTick()
    {
        if (start == true)
            timerbtn.Text = DateTime.Now.ToString("T");
        else
            timerbtn.Text = "Таймер вимкнено";
        return start;
    }

    private void TimerButton_Clicked(object sender, EventArgs e)
    {
        if (start == true)
        {
            start = false;
        }
        else
        {
            start = true;
            Dispatcher.StartTimer(TimeSpan.FromSeconds(1), OnTimerTick);
        }
    }
}
```

Рисунок 1.1 – Приклад створення простого таймера в .NET MAUI на C#

Кнопка створюється як об'єкт типу **Button** з ініціалізацією властивостей через *object initializer*. Властивість *Text* встановлює початковий текст, який користувач бачить на кнопці. Властивості *VerticalOptions* та *HorizontalOptions* встановлюються в *LayoutOptions.Center*, що центрує кнопку на екрані за допомогою системи розміщення MAUI. Використання події *Clicked* через

оператор += пов'язує метод-обробник *TimerButton_Clicked* з подією натиску кнопки. Остаточно, створену кнопку присвоюємо властивості *Content* сторінки, що робить її видимою для користувача. Метод OnTimerTick() являє собою обробник, який викликається періодично через систему *Dispatcher* в MAUI. Цей метод повинен повертати логічне значення: якщо повертається true, таймер продовжує роботу; якщо false, таймер зупиняється. Це елегантна альтернатива до класичних подій таймера в Windows Forms.

Логіка методу проста: якщо таймер активний (змінна start == true), текст кнопки оновлюється на поточний час у форматі часу (годин:хвилин:секунд), отриманий з *DateTime.Now.ToString("T")*. Якщо таймер неактивний, на кнопці показується повідомлення «Таймер вимкнено». Цей метод демонструє важливу концепцію реактивного оновлення інтерфейсу – вміст екрана змінюється залежно від внутрішнього стану програми. Метод TimerButton_Clicked() викликається щоразу, коли користувач натискає кнопку. Це точка входу для управління станом таймера. Логіка реалізує простий механізм перемикання: якщо таймер уже запущений (start == true), він зупиняється присвоюванням start = false. Якщо таймер зупинений, він активується присвоюванням start = true та запуском періодичного виклику.

Ключова операція – Dispatcher.StartTimer(TimeSpan.FromSeconds(1), OnTimerTick). Вона запускає системний таймер, який викликає метод *OnTimerTick* кожну секунду (визначено через TimeSpan.FromSeconds(1)).

1.2 Використання Task.Delay() для реалізації асинхронних часових затримок

Метод Task.Delay() є однією з найбільш універсальних і широко використовуваних функцій в асинхронному програмуванні на платформі .NET MAUI. На відміну від застарілого підходу Thread.Sleep(), який блокує поточний потік виконання і призводить до заморозки користувацького інтерфейсу, метод Task.Delay() повертає завершену задачу (**Task**), яка

дозволяє викликаючому коду продовжити виконання без блокування потоку UI. Це є критично важливим для розробки кросплатформових застосунків, оскільки .NET MAUI розраховує на асинхронну обробку операцій для підтримки гладкої роботи інтерфейсу на всіх платформах (iOS, Android, macOS, Windows).

Метод є перевантаженим і пропонує чотири основні варіанти для роботи з часовими затримками: *Task.Delay(Int32)* – найпростіша форма методу, яка приймає кількість мілісекунд як параметр типу Int32. При виклику повертає Task, який буде завершений після закінчення вказаного часу. Ця перевантаженість застосовується в простих сценаріях, де не потрібна можливість відміни операції. *Task.Delay(Int32, CancellationToken)* – надає механізм для раннього припинення очікування за допомогою CancellationToken. Цей параметр дозволяє зовнішньому коду запросити скасування операції, що є неоціненним у сценаріях, де користувач повинен мати можливість перервати тривалу операцію (наприклад, зупинити таймер, скасувати завантаження або припинити моніторинг). *Task.Delay(TimeSpan)* – альтернативна форма, яка замість мілісекунд приймає структуру TimeSpan. Така форма більш читаема і зручна під час роботи з більшими часовими проміжками, оскільки дозволяє явно вказати часові одиниці (години, хвилини, секунди, мілісекунди), підвищуючи наочність коду. *Task.Delay(TimeSpan, CancellationToken)* – комбінує переваги обох попередніх варіантів, надаючи як зручне представлення часу через *TimeSpan*, так і механізм скасування через *CancellationToken*.

2 РОБОЧЕ ЗАВДАННЯ

Завдання 1. Проаналізувати теоретичні основи розробки кросплатформових додатків за допомогою .NET MAUI, викладені в розділі 1 навчального матеріалу. На основі отриманих знань дослідити функціональні можливості інтегрованого середовища розробки Visual Studio, зокрема інструменти для створення таймерів у MAUI-додатках. Практично реалізувати

всі наведені у розділі приклади, забезпечивши їх працездатність на принаймні одній цільовій платформі (Windows, Android або iOS).

Завдання 2. Розробити кросплатформовий додаток за допомогою технології .NET MAUI, головне вікно якого містить прямокутник, розміщений у центрі екрана. Прямокутник повинен періодично змінювати свій колір і розмір. Кольори, між якими здійснюється перемикання, слід задати у вигляді масиву. Перехід між кольорами має відбуватися послідовно або циклічно (після останнього кольору відновлюється перший). У додатку потрібно реалізувати елемент керування **Slider**, який дозволяє користувачеві змінювати частоту (періодичність) зміни кольору прямокутника. Значення **Slider** повинно впливати на інтервал роботи таймера, що здійснює оновлення параметрів об’єкта.

3 КОНТРОЛЬНІ ПИТАННЯ

1. Що таке таймер у контексті платформи .NET MAUI?
2. У яких ситуаціях доцільно використовувати таймер у мобільних додатках?
3. Які існують основні способи створення таймерів у.NET MAUI?
4. Як реалізується таймер за допомогою методу Dispatcher.StartTimer()?
5. Як організувати роботу таймера із використанням асинхронного методу Task.Delay()?

ПРАКТИЧНА РОБОТА №7

Тема роботи: дослідження можливостей Visual Studio та .NET MAUI для створення додатків з таблицями.

Мета роботи: отримати практичні навички для створення і використання таблиць в середовищі Visual Studio з використанням .NET MAUI.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Елемент **TableView**

Елемент **TableView** у .NET MAUI – це потужний компонент для побудови структурованих, багатосекційних форм і налаштувань на мобільних і десктопних платформах, що дозволяє реалізувати зручні інтерфейси для введення, перегляду і редагування даних, а також створювати інтуїтивні меню та конфігураційні списки. **TableView** побудований на принципі ієрархії контейнерів і осередків (*Cell*), що відповідає підходам сучасної мобільної розробки. Кореневим елементом є *TableRoot*, який містить одну або декілька секцій *TableSection*. Секції, у свою чергу, включають набір осередків різних типів – *EntryCell* для текстового вводу, *SwitchCell* для перемикачів, *TextCell* для простих текстових елементів, *ImageCell* для елементів із зображенням тощо. Така гнучка організація дозволяє конструювати як прості списки, так і складні багаторівневі форми з різноманітними елементами взаємодії. Важливо, що у MAUI **TableView** надає можливість комбінувати різні типи осередків у межах однієї таблиці, що особливо корисно для сторінок налаштувань або багатокомпонентних форм. Наприклад, секція персональних даних може містити текстові поля для вводу і перемикач для збереження інформації, а секція контактів – поля для номера телефону та електронної пошти. Основні властивості **TableView**: *HasUnevenRows*: Дозволяє задавати таблиці змінну висоту рядків, що актуально для осередків із великою

кількістю інформації або зображеннями. *Intent*: Визначає стилі і логіку використання таблиці за допомогою перерахування *TableIntent* (*Data, Form, Menu, Setting*). Це дає змогу явно вказати контекст застосування – чи це форма для введення, список даних, меню вибору або сторінка налаштувань. *Root*: Корінь таблиці, іменований *TableRoot*, дозволяє зручно організувати і управляти вмістом *TableView. RowHeight*: Дозволяє встановити фіксовану висоту рядків для рівномірних таблиць, але якщо *HasUnevenRows* істинно – висота визначається автоматично залежно від вмісту.

TableView у MAUI може бути оголошено як у XAML (декларативно) (рис. 1.1), так і у C# (програмно) (рис. 1.2), залежно від завдань і архітектури додатка.

```
<TableView>
    <TableView.Root>
        <TableRoot Title="Ввод данных">
            <TableSection Title="Персональні дані">
                <EntryCell Label="Логін" Keyboard="Default" Placeholder="Введіть логін"/>
                <SwitchCell Text="Зберегти"/>
            </TableSection>
            <TableSection Title="Контакти">
                <EntryCell Label="Телефон" Keyboard="Telephone" Placeholder="Номер телефону"/>
                <EntryCell Label="Email" Keyboard="Email" Placeholder="Введіть email"/>
            </TableSection>
        </TableRoot>
    </TableView.Root>
</TableView>
```

Рисунок 1.1 – Створення елементу **TableView** у XAML

Такий підхід дає змогу гнучко описувати структуру таблиці і легко змінювати її через XAML-розмітку, що спрощує рефакторинг та підтримку. TableView у MAUI є стандартом для реалізації сторінок налаштувань, форм реєстрації, анкет користувача, контактної інформації чи списків процедур і параметрів.

```
this.Content = new TableView
{
    Intent = TableIntent.Form,
    Root = new TableRoot("Ввод данных")
    {
        new TableSection("Персональні дані")
        {
            new EntryCell
            {
                Label = "Логін:",
                Placeholder = "введіть логін",
                Keyboard = Keyboard.Default
            },
            new SwitchCell { Text = "Зберегти" }
        },
        new TableSection("Контакти")
        {
            new EntryCell
            {
                Label = "Телефон:",
                Placeholder = "введіть телефон",
                Keyboard = Keyboard.Telephone
            },
            new EntryCell
            {
                Label = "Email:",
                Placeholder = "введіть email",
                Keyboard = Keyboard.Email
            }
        }
    }
};
```

Рисунок 1.2 – Створення елементу **TableView** у C#

1.2 Клас *Cell*

Клас *Cell* у .NET MAUI є абстрактним базовим елементом ієрархії елементів списків, який задає спільну поведінку, життєвий цикл і контракт взаємодії для всіх типів клітинок, що вбудовуються в **ListView** та **TableView**, а також у контейнери колекцій зі зворотною сумісністю після міграції з Xamarin.Forms. Клітинки самі по собі не є повноцінними візуальними елементами, а радше полегшеними представниками даних, які MAUI матеріалізує у нативні осередки списку на рівні платформи, керуючи їхнім

відтворенням, станом і подіями відповідно до життєвого циклу спискових інструментів. Така абстракція дає змогу описувати модель клітинки в XAML або C#, відділяючи декларацію даних і команд від платформної реалізації, що відповідає загальній архітектурі MAUI «єдине API поверх нативних платформ».

Властивість *ContextActions* надає колекцію елементів контекстного меню, які відображаються при виконанні користувачем платформозалежного жесту (наприклад, свайп або довге натискання) над клітинкою, що дозволяє додавати дії на кшталт «Видалити», «Редагувати» без зміни макета клітинки. Похідна властивість *HasContextActions* повертає ознаку наявності хоча б одного пункту в колекції *ContextActions*, спрощуючи умови відображення та логіку ввімкнення жестів або декоративних індикаторів для клітинок із діями. Властивість *Height* встановлює або повертає бажану висоту клітинки, що використовується механізмом вимірювання для побудови списку, водночас *RenderHeight* надає обчислену фактичну висоту після розміщення на цільовому пристрої з урахуванням щільності пікселів та платформних правил компонування. Властивість *IsEnabled* керує доступністю клітинки до взаємодії та прив’язується до стану команд і джерела даних, формально визначаючи, чи клітинка може реагувати на події та чи застосовуються до неї стилі станів.

Життєвий цикл клітинки експонується через події *Appearing* і *Disappearing*, що сигналізують про додавання або вилучення візуального представлення клітинки з ієрархії відображення, дозволяючи ініціалізувати ресурси, підписки або звільняти їх у момент фактичної візуалізації чи рециклінгу елемента списку. Подія *Tapped* виникає при натисканні на клітинку і слугує універсальним входом для обробки вибору або запуску команд без необхідності підписуватися на події конкретних підпредставлень усередині клітинки. У типовій MVVM-побудові ці події корелюють з командами, а зв’язування контексту (*BindingContext*) клітинки здійснюється через механізми MAUI, що забезпечує передачу елементів даних та команд на рівень уявлення без платформної логіки, згідно з принципами єдиного коду в MAUI.

1.2.1 Об’єкт **EntryCell**

EntryCell у .NET MAUI історично означає клітинку з міткою та однорядковим полем введення в складі **TableView**, але в актуальних версіях MAUI ці API позначені як застарілі й рекомендується перехід на сучасніші елементи, насамперед *CollectionView* із власними елементами введення на базі Entry або повноцінні форми без **TableView**. **EntryCell** – це тип *Cell*, що поєднує статичну мітку та однорядковий редактор тексту, і використовується виключно всередині **TableView** чи **ListView** для побудови формоподібних екранів налаштувань або простих форм. В .NET 10 для .NET MAUI сімейство Cell (**EntryCell, TextCell, ImageCell, SwitchCell, ViewCell**), разом із **ListView** і **TableView**, офіційно позначено застарілим; рекомендовано мігрувати до **CollectionView** або складати форми звичайними розмітками (*Grid/VerticalStackLayout*) із **Entry**, що дає кращу гнучкість і продуктивність.

Ключові властивості **EntryCell:** *HorizontalTextAlignment*: горизонтальне вирівнювання тексту поля введення в межах **EntryCell**, доступне як bindable-властивість для керування композицією тексту. *Keyboard*: тип клавіатури, що з’являється під час редагування (Default, Numeric тощо); важливо врахувати, що на iOS числова клавіатура може не мати кнопки «Done», що вплине на сценарії обробки завершення введення. *Label*: фіксований підпис поруч із полем введення, який слугує контекстною підказкою щодо призначення значення. *LabelColor*: колір візуалізації підпису; застосовується до статичної частини клітинки з урахуванням стилів/тем. *Placeholder*: текст-заповнювач у полі вводу, що відображається, коли значення порожнє або null, полегшуючи розуміння очікуваного формату. *Text*: основний вміст поля введення; як правило, прив’язується до властивостей *ViewModel* для двосторонньої синхронізації. *VerticalTextAlignment*: вертикальне вирівнювання тексту поля (не завжди згадуване в оглядах, але доступне як bindable-властивість).

Головна подія – *Completed*: вона спрацьовує, коли користувач підтверджує введення клавішею «Return/Done» на клавіатурі, що доречно для валідації або переходу до наступного поля. На iOS із Keyboard=«Numeric»

кнопки «Done» немає, тому «Completed» може не викликатися; у таких сценаріях застосовують інші шаблони (кастомні аксесуари клавіатури або альтернативні події/розкладки) замість покладання лише на «Completed».

```
var header = new Label
{
    Text = "EntryCell",
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    HorizontalOptions = LayoutOptions.Center
};

var tableView = new TableView
{
    Intent = TableIntent.Form,
    Root = new TableRoot
    {
        new TableSection
        {
            new EntryCell
            {
                Label = "EntryCell:",
                Placeholder = "Type text here"
            }
        }
    }
};

Content = new StackLayout
{
    Children = { header, tableView }
};
```

Рисунок 1.3 – Розміщення елементу **EntryCell** у **TableView** у C#

1.2.2 Об’єкт **SwitchCell**

Об’єкт **SwitchCell** у .NET MAUI є спеціалізованим елементом управління, який використовується в межах компонента **TableView**. Він поєднує текстову мітку (**Label**) і перемикач стану (**Switch**). Така комірка призначена для відображення налаштувань, що можуть бути активовані або вимкнені користувачем, наприклад: «Увімкнути сповіщення», «Дозволити

GPS», «Режим енергозбереження» тощо. **SwitchCell** слугує частиною інтерфейсу користувача, де необхідно надати зручний спосіб перемикання логічних параметрів типу true/false. На відміну від автономного елемента **Switch**, який можна розміщувати будь-де в макеті, **SwitchCell** вбудовується безпосередньо у таблиці (**TableView**) і автоматично забезпечує відповідне оформлення, відступи та поведінку, характерну для списків налаштувань у мобільних додатках. Основні властивості **SwitchCell**: *Text* – задає або повертає текстову мітку, розташовану ліворуч від перемикача. Це може бути коротке пояснення чи назва параметра, який користувач змінює. *On* – визначає логічний стан перемикача. Приймає значення true (увімкнено) або false (вимкнено). Це основна властивість, що відображає поточний стан опції.

Основна подія об’єкту **SwitchCell**: *OnChanged* – подія, яка виникає щоразу, коли користувач змінює стан перемикача. Вона дозволяє обробляти зміни, наприклад, оновлювати налаштування програми або зберігати вибір користувача у базі даних чи локальному сховищі. Елемент **SwitchCell** використовується всередині колекції **TableSection**, яка, у свою чергу, входить до складу **TableRoot** компонента **TableView**. Така ієрархія дозволяє створювати групи пов’язаних налаштувань, розділених на логічні секції.

Нижче наведено приклад створення сторінки, яка містить заголовок та таблицю з одним елементом **SwitchCell (**рис. 1.4**).** У цьому прикладі створюється сторінка з одним перемикачем «Увімкнути сповіщення». Подібна форма часто використовується у розділах налаштувань мобільних застосунків. Подальше підключення події *OnChanged* дозволяє реагувати на зміну стану, наприклад, відключати або активувати фонові служби, змінювати колірну тему тощо.

```
public class SwitchCellPage : ContentPage
{
    public SwitchCellPage()
    {
        Label header = new Label
        {
            Text = "SwitchCell",
            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
            HorizontalOptions = LayoutOptions.Center
        };
        TableView tableView = new TableView
        {
            Intent = TableIntent.Form,
            Root = new TableRoot
            {
                new TableSection("Налаштування")
                {
                    new SwitchCell
                    {
                        Text = "Увімкнути сповіщення",
                        On = true
                    }
                }
            }
        };
        Content = new StackLayout
        {
            Children = { header, tableView }
        };
    }
}
```

Рисунок 1.4 – Розміщення об’єкту **SwitchCell** у **TableView** у C#

1.2.3 Робота з об’єктами **TextCell**

Об’єкт **TextCell** у .NET MAUI використовується для створення простих відображень текстової інформації у списках або таблицях, коли немає потреби у складній розмітці. Кожен елемент **TextCell** складається з двох текстових полів: основного (*Text*) і додаткового (*Detail*). Такий тип комірки зручний для відображення коротких пар даних, наприклад, назви параметра і його значення, імені контакту та номера телефону, чи заголовку та короткого опису. На відміну від звичайних елементів, побудованих за допомогою **Label** або **StackLayout**, **TextCell** створюється спеціально для роботи в контейнерах типу **TableView** або **ListView**, які оптимізовані для представлення структурованих

наборів даних. Це дозволяє MAUI-інтерфейсу адаптивно відображати такі таблиці на різних платформах, дотримуючись системних стандартів оформлення (Android Material Design, iOS Human Interface Guidelines, Windows Fluent Design тощо). Основними властивостями об’єкта **TextCell** є: *Text* – основний текст елемента, що відображається більшим шрифтом. Використовується для виведення головної інформації або назви запису. *TextColor* – задає або повертає колір основного тексту. Може бути заданий через об’єкт **Color** (наприклад, *Colors.Black, Colors.Blue* тощо). *Detail* – додатковий текст, що з’являється під основним. Його застосування доречне для уточнення чи пояснення значення, відображеного у полі *Text*. *DetailColor* – визначає колір додаткового тексту. Як правило, використовується сірий або приглушений відтінок, щоб візуально підкреслити вторинність цих даних.

У MAUI логіка побудови інтерфейсу передбачає використання декларативного XAML або програмного опису на C#. Нижче наведено приклад коду мовою C# (рис. 1.5), який демонструє створення **TableView** з елементом **TextCell**. У цьому прикладі створюється заголовок **Label**, після чого конструюється об’єкт **TableView** із розділом **TableSection**, який містить один елемент **TextCell**. Властивості *Text* та *Detail* задають зміст, а параметри *TextColor* і *DetailColor* визначають стиль візуального відображення. Завдяки розміщенню у контейнері **StackLayout**, таблиця і заголовок утворюють вертикальну структуру сторінки.

Важливо зазначити, що *TextCell* не є самостійним візуальним елементом у контексті сторінки. Його можна використовувати виключно в межах *TableView*, *ListView* або *CollectionView*, оскільки рендерінг і поведінка клітинки залежать від контейнера. Коли застосунок виконуватиметься на різних платформах, *TextCell* автоматично підлаштовуватиме розмітку під нативну систему, зберігаючи узгоджений користувацький досвід.

```
Label header = new Label
{
    Text = "TextCell приклад",
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    HorizontalOptions = LayoutOptions.Center
};
TableView tableView = new TableView
{
    Intent = TableIntent.Form,
    Root = new TableRoot
    {
        new TableSection("Приклад розділу")
        {
            new TextCell
            {
                Text = "Назва елемента",
                Detail = "Додатковий опис",
                TextColor = Colors.Black,
                DetailColor = Colors.Gray
            }
        }
    }
};
Content = new StackLayout
{
    Children = { header, tableView }
};
```

Рисунок 1.5 – Розміщення об’єкту **TextCell** у **TableView** у C#

1.2.4 Об’єкт **ImageCell**

Компонент **ImageCell** у .NET MAUI призначений для відображення комбінації графічного зображення та текстової інформації в межах елемента списку або таблиці. Клас **ImageCell** є похідним від базового класу Cell і зазвичай використовується у колекційних представленнях, наприклад, **TableView** або **ListView**. Такий осередок зручний для відображення структурованих даних, де візуальний контент (наприклад, ескіз, іконка чи фото) поєднується з коротким описом або деталями. Основні властивості: *ImageSource* – визначає джерело зображення, яке буде завантажено та відображено в осередку. Джерелом може бути локальний файл (наприклад, ресурс проєкту у вигляді файлу .png чи .jpg), URI-адреса (завантаження з інтернету), або об’єкт типу *StreamImageSource*, якщо зображення формується динамічно. *Text* – головний підпис, який розташовується поруч із

зображенням. Його зазвичай використовують для короткого опису сутності. *Detail* – допоміжний текст, що відображається під основним заголовком. Може містити додаткову інформацію чи уточнення. *ImageSourceAspect* (починаючи з .NET MAUI) – задає спосіб масштабування зображення (наприклад, *AspectFit, AspectFill, Fill*), що дозволяє контролювати, як картинка вміщується в межах комірки.

Використання **ImageCell** дозволяє створювати таблиці або списки з привабливим змістом, де елементи містять як графіку, так і текст. Наприклад, у застосунку можна відображати список автомобілів (рис. 1.6), користувачів чи товарів із невеликим зображенням ліворуч і підписами праворуч.

```
Label header = new Label
{
    Text = "Демонстрація ImageCell",
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
    HorizontalOptions = LayoutOptions.Center
};
TableView tableView = new TableView
{
    Intent = TableIntent.Form,
    Root = new TableRoot
    {
        new TableSection("Приклад секції")
        {
            new ImageCell
            {
                ImageSource = "car1.jpg",
                Text = "Tesla Model S",
                Detail = "Повністю електричний автомобіль"
            },
            new ImageCell
            {
                ImageSource = "car2.jpg",
                Text = "BMW i4",
                Detail = "Електричне купе головного класу"
            }
        }
    }
};
this.Content = new StackLayout
{
    Children = { header, tableView }
};
```

Рисунок 1.6 – Розміщення об’єкту **ImageCell** у **TableView** у C#

У цьому прикладі створюється заголовок та таблиця типу **TableView**, що містить секцію з двома **ImageCell**. Кожна комірка поєднує зображення транспортного засобу, основну назву та додатковий опис. Елементи автоматично впорядковуються і підлаштовуються під платформу, забезпечуючи нативний вигляд інтерфейсу.

1.2.6 Об’єкт **ViewCell**

У .NET MAUI, коли стандартних клітинок (**TextCell, ImageCell, SwitchCell, EntryCell**) для відображення та збору даних у таблицях або списках недостатньо, розробник може створити користувацькі клітинки за допомогою класу **ViewCell**. **ViewCell** є базовим типом клітинки, зовнішній вигляд якої повністю визначається вкладеним елементом **View**. Це дає змогу реалізувати практично будь-який інтерфейс всередині одного рядка таблиці або списку. У випадку **TableView**, який у .NET MAUI використовується для роботи з відносно статичними або вручну змінюваними даними, немає концепції шаблону елемента як у **ListView** або **CollectionView**. Тому користувацькі клітинки створюються вручну і додаються до секцій таблиці. Для повної кастомізації вмісту клітинки застосовується **ViewCell**, в якому всередині можна використовувати будь-які layout-компоненти – **StackLayout**, **Grid**, **FlexLayout** тощо. **ViewCell** має властивість *View*, що вказує на віджет, який відображатиме вміст клітинки. Це може бути контейнер, що містить кілька елементів керування, наприклад, **Label, Image, Button, Switch** тощо. Прикладом може слугувати горизонтально орієнтований **StackLayout**, який розміщує текстову мітку, зображення, кнопку й перемикач поруч. Приклад використання користувацьких клітинок **ViewCell** показаний на рисунку 1.7 на мові XAML і на мові C# – на рисунку 1.8. У цьому прикладі **StackLayout** відповідає за розташування елементів всередині клітинки по горизонталі. Кожен елемент може мати свої властивості, стилі, події тощо.

```xaml
<TableView Intent="Settings">
    <TableRoot>
        <TableSection Title="Користувацька клітинка">
            <ViewCell>
                <StackLayout Orientation="Horizontal">
                    <Label Text="Мітка" TextColor="#f35e20" />
                    <Image Source="BMW X4.jpg" />
                    <Button Text="Кнопка" />
                    <Switch HorizontalOptions="Center" />
                </StackLayout>
            </ViewCell>
        </TableSection>
    </TableRoot>
</TableView>
```

Рисунок 1.7 – Розміщення об’єкту **ViewCell** у **TableView** у XAML

```csharp
var table = new TableView();
var layout = new StackLayout { Orientation = StackOrientation.Horizontal };

layout.Children.Add(new Label { Text = "Мітка", TextColor = Color.FromHex("#f35e20") });
layout.Children.Add(new Image { Source = "BMW X4.jpg" });
layout.Children.Add(new Button { Text = "Кнопка" });
layout.Children.Add(new Switch { HorizontalOptions = LayoutOptions.Center });

table.Root = new TableRoot
{
    new TableSection("[translate:Користувацька клітинка]")
    {
        new ViewCell { View = layout }
    }
};
Content = table;
```

Рисунок 1.8 – Розміщення об’єкту **ViewCell** у **TableView** у C#

2 РОБОЧЕ ЗАВДАННЯ

Завдання 1. Проаналізувати теоретичні основи розробки кросплатформових додатків за допомогою .NET MAUI, викладені в розділі 1 навчального матеріалу. На основі отриманих знань дослідити функціональні можливості інтегрованого середовища розробки Visual Studio, зокрема

інструменти для створення таблиць у MAUI-додатках. Практично реалізувати всі наведені у розділі приклади, забезпечивши їх працездатність на принаймні одній цільовій платформі (Windows, Android або iOS).

Завдання 2. Необхідно створити та записати програмний код реалізації прикладів, що показані на рисунках 1.3, 1.4, 1.5 та 1.6 на мові XAML.

3 КОНТРОЛЬНІ ПИТАННЯ

1. Яке призначення елемента **TableView** у .NET MAUI?
2. Які основні властивості має елемент **TableView**?
3. Яке призначення елемента **Cell** у структурі **TableView**?
4. Які основні властивості визначають поведінку елемента **Cell**?
5. Яке призначення елемента **EntryCell** у розмітці інтерфейсу?
6. Які основні властивості характерні для елемента **EntryCell**?
7. Яке призначення елемента **SwitchCell** у **TableView**?
8. Які основні властивості має елемент **SwitchCell**?
9. Яке призначення елемента **TextCell** у побудові інтерфейсу?
10. Які основні властивості притаманні елементу **TextCell**?
11. Яке призначення елемента **ImageCell** у візуальному відображенні даних?
12. Які основні властивості використовуються для налаштування елемента **ImageCell**?

ПРАКТИЧНА РОБОТА №8

Тема роботи: дослідження можливостей Visual Studio та .NET MAUI для створення додатків з вкладками.

Мета роботи: отримати практичні навички для створення і використання додатків з вкладками в середовищі Visual Studio з використанням .NET MAUI.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Створення сторінок з вкладками за допомогою мови XAML

У технології .NET MAUI для кросплатформної розробки застосовується елемент управління **TabbedPage**, який дозволяє створювати сторінки з вкладками для зручної навігації між різними розділами або функціональними екранними областями програми. **TabbedPage** є контейнером, що підтримує колекцію дочірніх сторінок (зазвичай **ContentPage**), лише одна з яких одночасно відображається користувачеві. Користувач може переключатися між ними за допомогою вкладок, розташованих на панелі у верхній або нижній частині екрана, залежно від платформи. На iOS вкладки традиційно розташовуються внизу екрана, що відповідає інтерфейсним стандартам цієї платформи, а основний контент сторінки – над вкладками. На пристроях Android та Windows, навпаки, вкладки розміщуються вгорі екрану, а вміст сторінки – під ними. Якщо кількість вкладок перевищує ширину екрану, колекція вкладок у верхній частині може прокручуватися, дозволяючи зручно масштабувати інтерфейс на різних пристроях.

У порівнянні з Xamarin.Forms, у MAUI клас **TabbedPage** зберіг свою концепцію як набір сторінок-вкладок. Кожна вкладка визначається дочірньою сторінкою, яка реалізує контент конкретного розділу. **ContentPage** є базовим типом для кожної вкладки, на ній можна розмістити будь-які UI-елементи, реалізуючи бажаний інтерфейс. У MAUI для керування виглядом вкладок

доступні такі властивості **TabbedPage**, як *BackgroundColor* (фон сторінки), *BarBackgroundColor* (фон панелі вкладок), *BarTextColor* (колір тексту на панелі), а також *SelectedTabColor* та *UnselectedTabColor*, що дозволяють гнучко керувати стилем вкладок. Заголовок вкладки задається через властивість *Title* сторінки, а іконка – властивістю *IconImageSource*.

Для створення сторінок з вкладками в MAUI використовують XAML, де головна сторінка змінюється з **ContentPage** на **TabbedPage**, а в колекцію **Children** додаються дочірні **ContentPage**, які стануть вкладками. Ось приклад базової структури XAML із трьома сторінками-вкладками (рис. 1.1).

```
<TabbedPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            xmlns:local="clr-namespace:FirstApp"
            x:Class="FirstApp.MainPage">
    <local:Page1 />
    <local:Page2 />
    <local:Page3 />
</TabbedPage>
```

Рисунок 1.1 – Створення вкладки у XAML

Тут Page1, Page2 та Page3 – це **ContentPage**, кожна з яких має власний *Title*. Код сторінок у файлах «.xaml.cs» традиційно залишається без змін або містить мінімальний необхідний код ініціалізації. Головна сторінка проекту (**MainPage**) у коді C# (рис. 1.2) успадковує **TabbedPage**:

```
public partial class MainPage : TabbedPage
{public MainPage(){InitializeComponent();}}
```

Рисунок 1.2 – Створення вкладки у C#

Після запуску застосунку ця головна сторінка відобразить інтерфейс із вкладками Page1, Page2, Page3, між якими користувач зможе перемикатися.

1.2 Створення сторінок з вкладками за допомогою шаблону **TabbedPage**

TabbedPage є спеціальним типом сторінки в .NET MAUI, який дає змогу організувати навігацію через вкладки. Кожна вкладка представляє одну дочірню сторінку (зазвичай **ContentPage**), і користувач може перемикатися між ними за допомогою вкладок, які відображаються у вигляді панелі над чи під вмістом сторінки залежно від платформи. Наприклад, на iOS вкладки розташовуються внизу екрана, а на Android і Windows – у верхній частині. **TabbedPage** утримує колекцію дочірніх сторінок (children), з яких одночасно виводиться лише одна. Кожна дочірнвя сторінка має своє заголовок (*Title*), який відображається на вкладці, і, опційно, іконку (*IconImageSource*). Ця структура дозволяє зручно організовувати інтерфейс, коли потрібно розділити функціонал на кілька логічних частин із швидким перемиканням між ними.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="Primer2.TabbedPage1">
    <!-- Вкладки як дочірні сторінки -->
    <ContentPage Title="Tab 1" >
        <!-- Вміст сторінки вкладки 1 -->
    </ContentPage>
    <ContentPage Title="Tab 2" >
        <!-- Вміст сторінки вкладки 2 -->
    </ContentPage>
    <ContentPage Title="Tab 3" >
        <!-- Вміст сторінки вкладки 3 -->
    </ContentPage>
</TabbedPage>
```

Рисунок 1.3 – Створення сторінки із вкладками у XAML

Для створення сторінки з вкладками в проєкті MAUI рекомендується додати новий елемент типу **TabbedPage**. У XAML це може виглядати так (рис. 1.3). Тут кожна **ContentPage** – це окрема вкладка з власним заголовком. Вміст вкладок задається або безпосередньо у XAML, або програмно (рис. 1.4).

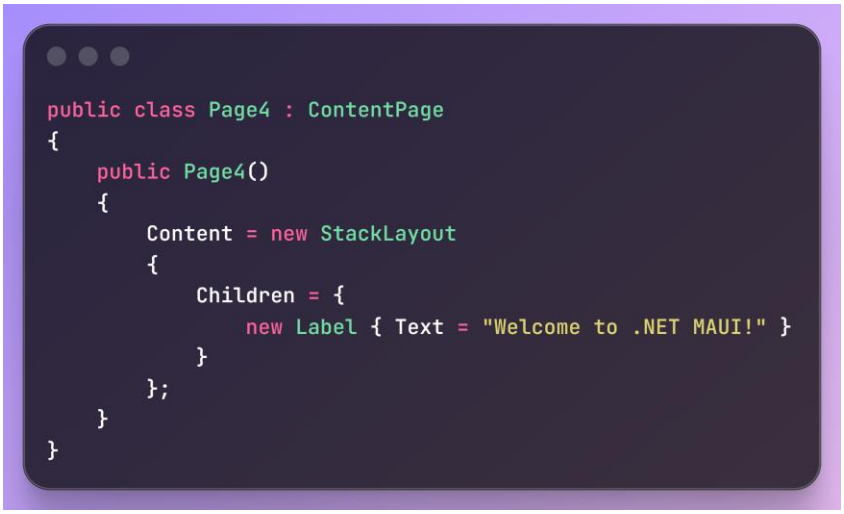
```
using Microsoft.Maui.Controls;

namespace Pryklad2
{
    public partial class TabbedPage1 : TabbedPage
    {
        public TabbedPage1()
        {
            InitializeComponent();
        }
    }
}
```

Рисунок 1.4 – Ініціалізація компонента **TabbedPage** у C#

Щоб **TabbedPage** відображалась при запуску додатку, у файлі «App.xaml.cs» необхідно замінити встановлення головної сторінки: *MainPage = new TabbedPage1();*

1.3 Створення сторінок з вкладками за допомогою мови C#

При створенні нового проекту знання шаблону «Forms Blank Content Page (C#)» дозволяє автоматично додавати код нової сторінки у вигляді окремого файлу класу. Приклад такої сторінки виглядає як клас, що наслідує **ContentPage** з базовим вмістом, наприклад (рис. 1.5).



```
public class Page4 : ContentPage
{
    public Page4()
    {
        Content = new StackLayout
        {
            Children = {
                new Label { Text = "Welcome to .NET MAUI!" }
            }
        };
    }
}
```

Рисунок 1.5 – Приклад створення сторінки **ContentPage** на C#

Для перетворення цієї **ContentPage** у вкладку, початковий клас потрібно змінити: натомість наслідувати **TabbedPage**, а всередині конструктора додати дочірні сторінки за допомогою властивості *Children*. Наприклад, щоб у вкладці відобразити центрально розташований червоний прямокутник розміром 50x100 пікселів, код виглядатиме так (рис. 1.6).

```
public class Page4 : TabbedPage
{
    public Page4()
    {
        this.Title = "Page4";

        this.Children.Add(new ContentPage
        {
            Content = new BoxView
            {
                Color = Colors.Red,
                HeightRequest = 100,
                WidthRequest = 50,
                VerticalOptions = LayoutOptions.Center,
                HorizontalOptions = LayoutOptions.Center
            }
        });
    }
}
```

Рисунок 1.6 – Відображення форматованого прямокутника **BoxView** у C#

Важливим кроком є додавання цієї вкладки до основної сторінки додатку, наприклад, у конструкторі MainPage: *this.Children.Add(new Page4());*

При запуску програми у локальному емуляторі або на мобільному пристрої інтерфейс відображатиме панель вкладок (зазвичай знизу на Android та зверху на iOS) зі списком доступних сторінок. Кожна вкладка динамічно завантажує свій контент при активації – це забезпечує оптимальне використання ресурсів пристрою, особливо важливе для мобільних платформ із обмеженою пам'яттю. Під час переходу користувача з одної вкладки на іншу MAUI автоматично управляє станом відповідних **ContentPage** об'єктів. У нашому прикладі при виборі «Page4» у центрі екрану відобразиться червоний прямокутник, а при виборі інших вкладок – їхній відповідний контент. Цей механізм забезпечує плавну навігацію та передбачуваний користувацький досвід.

2 РОБОЧЕ ЗАВДАННЯ

Завдання 1. Проаналізувати теоретичні основи розробки кросплатформових додатків за допомогою .NET MAUI, викладені в розділі 1 навчального матеріалу. На основі отриманих знань дослідити функціональні можливості інтегрованого середовища розробки Visual Studio, зокрема інструменти для створення вкладок у MAUI-додатках. Практично реалізувати всі наведені у розділі приклади, забезпечивши їх працездатність на принаймні одній цільовій платформі (Windows, Android або iOS).

Завдання 2. Необхідно модифікувати приклад на рисунку 1.3 таким чином, щоб колір вкладок був різним, а в центрі вкладки відображався надпис із номером вкладки, наприклад, «Вкладка №1».

Завдання 3. Необхідно розробити програму «Zavdannya3», яка відображає сторінку з трьома вкладками. Кожна вкладка має містити відповідний тип інформації про автомобіль.

Перша вкладка: містить зображення автомобіля.

Друга вкладка: відображає текстову інформацію з назвою моделі автомобіля та даними про виробника. Наприклад:

BMW X5 (F15)

Виробник: BMW AG

Штаб-квартира: Німеччина, м. Мюнхен.

Третя вкладка: демонструє технічні характеристики автомобіля, наприклад:

Об’єм двигуна – 2993 см³

Потужність двигуна – 258 к.с.

Максимальна швидкість – 230 км/год

Після реалізації програми необхідно перевірити коректність роботи інтерфейсу в емуляторі або на реальному пристрої, забезпечивши зручне перемикання між вкладками.

3 КОНТРОЛЬНІ ПИТАННЯ

1. Яке призначення елемента **TabbedPage** у застосунках .NET MAUI?
2. Які основні властивості визначають поведінку та вигляд елемента **TabbedPage**?
3. Які існують способи створення сторінок з вкладками у .NET MAUI?
4. У якій послідовності виконується створення сторінки з вкладками під час використання окремих сторінок (**ContentPage**)?
5. Яка послідовність дій під час створення сторінки з вкладками на основі шаблону **TabbedPage**?
6. Яким чином можна додати нову вкладку до вже створеної сторінки з вкладками?

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What is .NET MAUI? - .NET MAUI. *Microsoft Learn: Build skills that open doors in your career.* URL: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-9.0 (date of access: 10.10.2025).

2. Introduction to .NET MAUI: A Complete Beginner's Guide. *eLuminous Technologies.* URL: https://eluminoustechnologies.com/blog/net-maui-guide/ (date of access: 05.10.2025).

3. Label - .NET MAUI. *Microsoft Learn: Build skills that open doors in your career.* URL: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/label?view=net-maui-10.0 (date of access: 07.10.2025).

4. Progress Telerik. .NET MAUI Entry Overview - Telerik UI for .NET MAUI. *Telerik & Kendo UI - .NET Components Suites & JavaScript UI Libraries*. URL: https://www.telerik.com/maui-ui/documentation/controls/entry/overview (date of access: 04.10.2025).

5. .NET MAUI Button Types | .NET Multi-platform App UI | DevExpress Documentation. *DevExpress Documentation*. URL: https://docs.devexpress.com/MAUI/404711/utility-controls/button/button-types (date of access: 07.10.2025).

6. Button - .NET MAUI. *Microsoft Learn: Build skills that open doors in your career.* URL: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/button?view=net-maui-9.0 (date of access: 08.10.2025).

7. Use Borders and Frames (Part 8 of 18). *Microsoft Learn: Build skills that open doors in your career.* URL: https://learn.microsoft.com/en-us/shows/building-apps-with-xaml-and-dotnet-maui/use-borders-and-frames-building-apps-with-xaml-and-dotnet-maui (date of access: 08.10.2025).

8. Easily Design a Group Box View in .NET MAUI | *Syncfusion Blogs. Syncfusion.* URL: https://www.syncfusion.com/blogs/post/group-box-view-in-dotnet-maui (date of access: 10.10.2025).

9. Image - .NET MAUI. *Microsoft Learn: Build skills that open doors in your career.* URL: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/image?view=net-maui-9.0 (date of access: 11.10.2025).

10. Cell Class (Microsoft.Maui.Controls). *Microsoft Learn: Build skills that open doors in your career.* URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.maui.controls.cell?view=net-maui-9.0 (date of access: 16.10.2025).

11. Sharma A. K. TabbedPage in .NET MAUI. C# Corner: *AI-Powered Upskilling and Growth Platform.* URL: https://www.c-sharpcorner.com/article/tabbedpage-in-net-maui/ (date of access: 14.10.2025).

12. TabbedPage - .NET MAUI. *Microsoft Learn: Build skills that open doors in your career.* URL: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/pages/tabbedpage?view=net-maui-9.0 (date of access: 13.10.2025).

13. Learning Mobile, Web, Desktop, and Cloud Development using .NET. *Modern .NET Tutorials & Certification 2025 | AI Coding As...* URL: https://www.dotnetschool.com/article/web-development/dot-net-maui/dot-net-maui-page-types-tabbedpage-flyoutpage-complete-guide (date of access: 13.10.2025).

14. Тімонін В. А. Кросплатформне програмування : навч.-методич. посіб. Харків: ХНАДУ, 2020. 96 с.