This presentation uses the Pretendard and D2Coding font.
Photo by Rubaitul Azad on Unsplash

.NET Conf 2024 x Seoul

Empowering Tomorrow - Full Stack, AI Enhanced, Developer Focused

Presented by aws elastic INFRAGISTICS Couchbase MVP .NET foundation

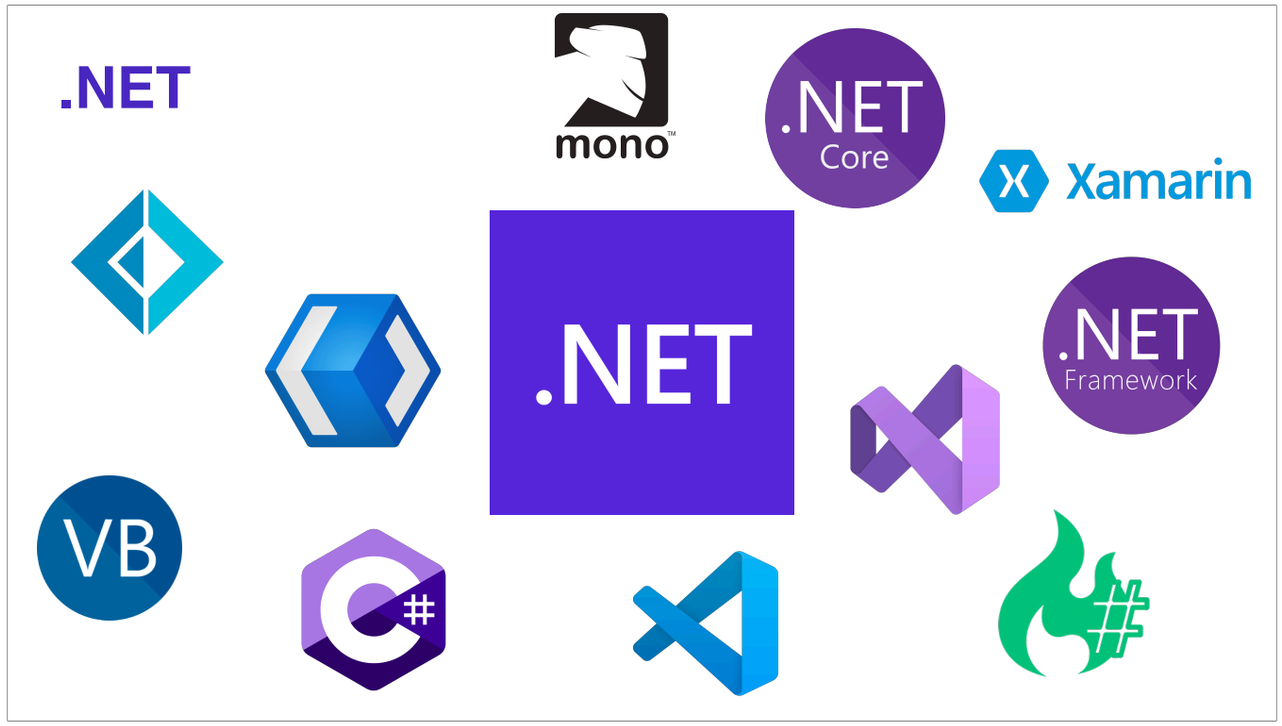
.Net Dev

안녕하세요! 저는 이번 발표를 진행하게 된 문예지, 릴리라고 합니다.
현재 썸아더플레이스에서 .NET으로 허그라는 iOS, Android 앱을 개발을 하고 있습니다.
사실 저는 불과 1년 전 만해도 .NET을 사용해 본적도 들어본 적도 없었습니다.
iOS 개발자를 목표로 구직을 하던 와중에 지금의 회사로부터 면접 제안을 받았고, 작년 5월부터 닷넷을 시작하게되었습니다,
닷넷을 처음 접했을 때 긴 역사와 함께 넓고 많은 기술들을 마주하며 닷넷으로 앱 개발을 한다는 게 어떤 그림인지 파악하기가 어려웠었습니다.
그래서 오늘 발표 내용는 닷넷을 좀 해보신 분들보다는 (닷넷을 처음 사용해보는) 닷린이분을 대상으로 닷넷으로 하는 크로스 플랫폼 모바일 앱 개발은 어떻게 진행되는지 공유해보려고 합니다.

발표 순서

- 사용 기술 스택 소개
  - NET
  - 개발 언어
  - Uno Platform
  - 개발 • 배포 도구
- Demo 앱 구현
  - Native API 사용
  - ViewModel 구현
  - UI 구현

발표 순서입니다
현재 .NET으로 앱을 만드는데 사용하는 기술 스택과 도구를 소개하고
간단한 MVVM 예제 앱을 만들어보며 구현 방법을 이야기해보겠습니다



.NET은 Microsoft에서 지원하는 오픈 소스 크로스플랫폼 개발 플랫폼인데요, Android, Apple, Linux, Windows 운영체제용 앱을 개발 할 수 있습니다.
여기서 개발 플랫폼이란 C#, F#, Visual Basic과 같은 언어, 비쥬얼스튜디오, 비쥬얼스튜디오코드와 같은 개발도구, 그리고 앱을 구축하기 위한 라이브러리, 프레임워크등을 모두 포함합니다.

싱글 코드로 다양한 운영체제, 아키텍처에서 동작하는 앱을 어떻게 만들 수 있을까요?
그것은 CLI가 있기 때문입니다. CLI는 Common Language Infrastructure의 줄임말로 다양한 고수준 언어들이 다양한 플랫폼, 아키텍처 마다 다시 쓰여질 필요 없도록 실행가능한 코드와 런타임 환경을 설명하는 기술적 표준입니다.
안드로이드와 아오에스는 이 CLI를 구현한 mono runtime에서 동작하게 됩니다.

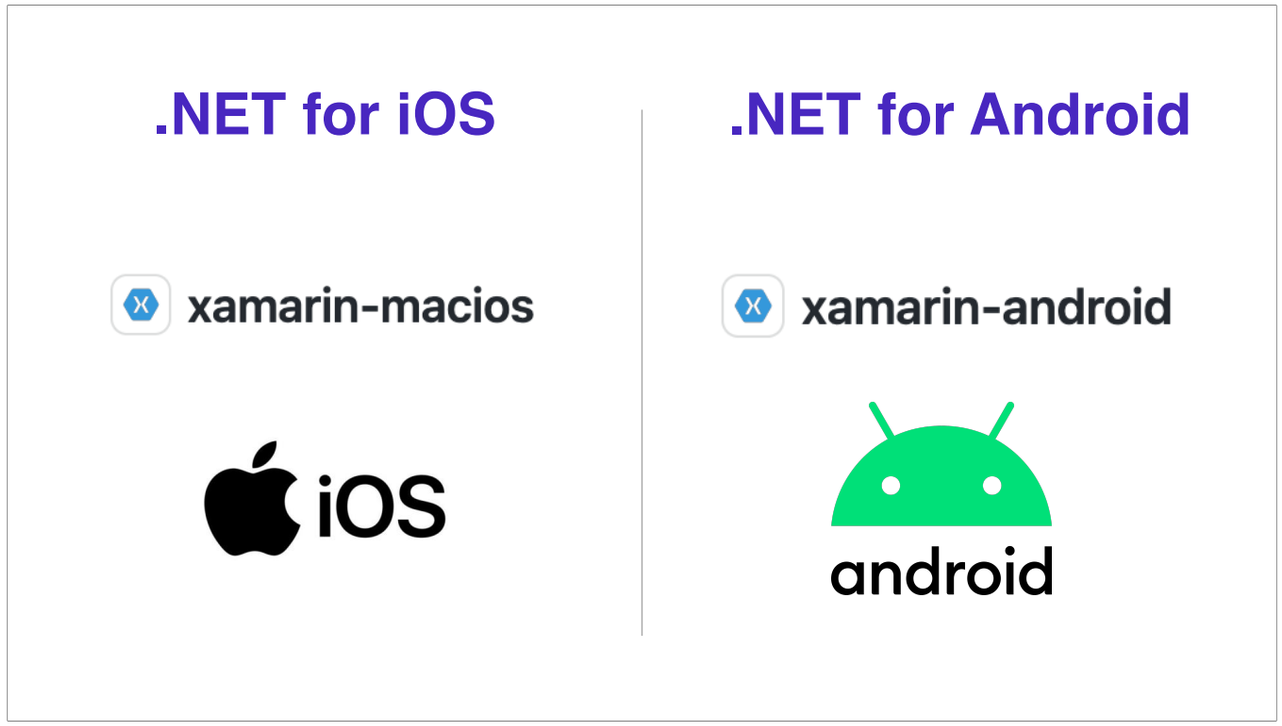
개발 언어

C#

Xaml

먼저 어떤 개발 언어를 사용할까요?

네이티브 iOS개발에선 Swift와 Objective-C를 사용하지만 닷넷에서는 C#과 Xaml을 사용합니다.
C# : .NET 프레임워크의 일부분으로 탄생한 언어로 객체지향언어의 특징을 가지고 있습니다. 앱 로직과 UI를 작성하는데 사용하고 있습니다.
Xaml : Extensible Application Markup Language의 줄임말로 선언형 태그 언어입니다. UI 트리를 구조적으로 쉽게 표현할 수 있는 장점이 있습니다. xaml로 작성된 UI 트리는 코드 비하인드인 C#파일에서도 동일하게 구현이 가능하며, UI와 로직을 분리하여 개발 할 수 있습니다.



.NET에는 모바일 앱 개발을 위한 프레임워크도 있습니다.
.NET for iOS/Android가 그거인데요, 이 두 프레임워크는 각각 iOS, 안드로이드 네이티브 API를 C#으로 매핑하여 네이티브 기술을 사용할 수 있도록 지원합니다
공식 로고가 없어서 깃헙 레포지토리의 대문 사진을 가지고 와봤는데요,
보시는 것과 같이 원래 Xamarin이라는 재단에서 Xamarin.iOS, Xamarin.Android로 개발해오던 프레임워크인데, 2016년 .NET이 인수하여 지금과 같은 이름으로 제공하고 있습니다.

GUI Framework

Xamarin

~ 2024.5.1

Xamarin.Forms

MAUI

Avalonia UI

Uno Platform

다음으로는 앱의 UI를 개발하는 .NET기반의 GUI Framework에 대해 알아보겠습니다
정말 다양한 종류의 프레임워크가 있지만, 오늘은 대표적인 4가지 프레임워크에 대해 간략하게 설명하고 넘어가겠습니다.

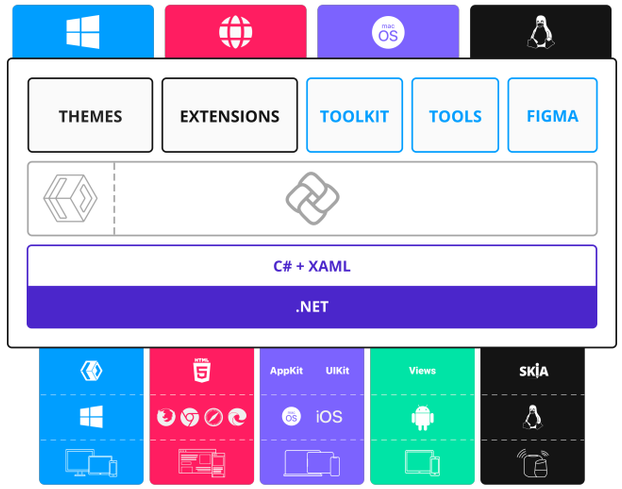
첫번째 Xamarin.Forms가 있습니다. 플랫폼 렌더러를 사용하여 단일 코드로 작성된 UI를 각 플랫폼의 네이티브 컨트롤로 변환하는 방식을 취합니다. 이는 2024년 5월부로 지원이 중단됩니다.

두번째 Maui입니다. Xamarin.Forms를 대체할 프레임워크로, 2022년 닷넷은 자마린을 기반으로 발전시킨 MAUI라는 프레임워크를 선보이게 됩니다. MAUI또한 네이티브 컨트롤로 변환하는 방식을 취합니다

저희 팀에선 처음에 MAUI를 사용해 마이그레이션을 진행햇지만, 출시된지 얼마 안된 프레임워크의 특성 상 버그와 불안정함 이슈로 MAUI 사용을 중단했었습니다.
세번째 AvaloniaUI 입니다. 앞서 말씀드린 자마린, 마우이와 다르게 네이티브 컨트롤로 변환하지 않고 렌더링 엔진을 사용해서 직접 픽셀로 UI를 그리는 방식을 취합니다.

마지막으로 Uno는 제가 현재 팀에서 사용하고 있는 프레임워크로, Skia를 사용한 직접 렌더링 방식과 네이티브 컨트롤 임베딩 방식 모두를 취하고 있습니다.

Uno Platform

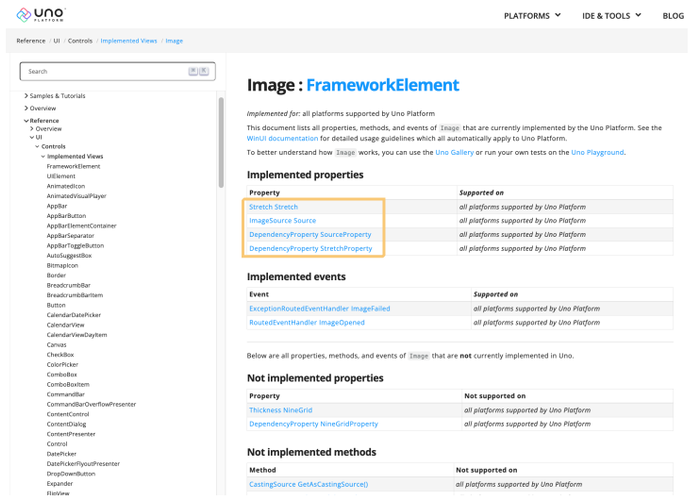


앞선 4개의 플랫폼들 중 저희가 사용하는 Uno Platform에 대해 좀 더 설명드리도록 하겠습니다.
iOS, Android뿐 아니라 Linux, Web, Window 용 앱도 개발하실 수 있습니다.
Uno는 앱의 UI와 Non-UI를 싱글 코드로 개발할 수 있는 프레임워크와 라이브러리들을 제공합니다

Uno Platform : UI

Source	이미지의 원본을 가져오거나 설정합니다.
SourceProperty	원본 종속성 속성을 식별합니다.
Stretch	대상 사각형을 채우기 위해 이미지를 확장해야 하는 방법을 설명하는 값을 가져오거나 설정합니다.
StretchProperty	Stretch 종속성 속성을 식별합니다.

WinUI - `Image`



Uno Platform - `Image`

Uno는 기존 WinUI를 사용하던 개발자들이 바로 모바일 앱 개발을 시작할 수 있도록 WinUI API를 네이티브 컨트롤 API에 매핑해두었습니다.
즉 WinUI 클래스와 프로퍼티를 그대로 가져다 쓰면, iOS, Android에서도 동일하게 보여지고 동작합니다.
예를 들어 FrameworkElement를 상속 받은 모든 뷰는 iOS의 모든 뷰 객체의 상위 클래스인 UIView를 상속 받습니다.
대부분의 Uno 컨트롤들은 실행시 iOS 네이티브 뷰로 변환되어 보여집니다.
다음과 같이 Uno 사이트에 들어가면 지원되는 API들을 확인할 수 있습니다. Uno개발을 고려하시는 분이 있으시다면 우노 플랫폼 도큐먼트가 잘 정리되어있으니 확인하시면 좋을 것 같습니다.

Uno Platform : Non UI

uno PLATFORM

PLATFORMS IDE & TOOLS BLOG DOC

Reference / Non-UI / Phone Calls

Search ⌘ K

- Samples & Tutorials
- Overview
- Reference
  - Overview
  - UI
  - Non-UI
    - Accelerometer
    - Android TV
    - App Actions
    - App Close Handler
    - App Suspension
    - Application Data and Settings
    - Badge Notifications
    - Barometer
    - Battery Information
    - Bluetooth
    - Clipboard
    - Compass
    - Contacts
    - Credential Storage
    - Device Enumeration
    - Device Information
    - E-mail
    - File Management
    - File and Folder Pickers
    - Flashlight
    - Gamepad
    - Geolocation (GPS)
    - Gyrometer
    - Haptics

Phone calls

TIP

This article covers Uno-specific information for `PhoneCallManager`. For a full description of the feature and instructions on using it, consult the UWP documentation: https://learn.microsoft.com/uwp/api/windows.applicationmodel.calls.phonecallmanager?view=winrt-22000

- `Windows.ApplicationModel.Calls.PhoneCallManager` allows apps to present a phone call UI to the user to allow them to initiate a call.
- The current call state can be checked using `IsCallActive` and `IsCallIncoming` properties.
- `CallStateChanged` event is raised when a call is incoming, accepted or dropped.

Supported features

Feature	Windows	Android	iOS	Web (WASM)	macOS	Linux (Skia)	Win 7 (Skia)
`ShowPhoneCallUI`	✔	✔	✔	✔	✖	✖	✖
`ShowPhoneCallSettingsUI`	✔	✔	✖	✖	✖	✖	✖
`CallStateChanged`	✔	✔	✔	✖	✖	✖	✖
`IsCallActive`	✔	✔	✔	✖	✖	✖	✖
`IsCallActive`	✔	✔	✔	✖	✖	✖	✖

Examples

Show phone call UI

또한 UI외 전화, 진동, 뱃지 노티피케이션과 같은 NON UI 피쳐도 싱글코드로 통합해서 제공됩니다.
하지만 아직 통합되지 않는 API들이 많아서 실무에서는 주로 .NET for iOS, Android를 사용합니다.

개발 도구

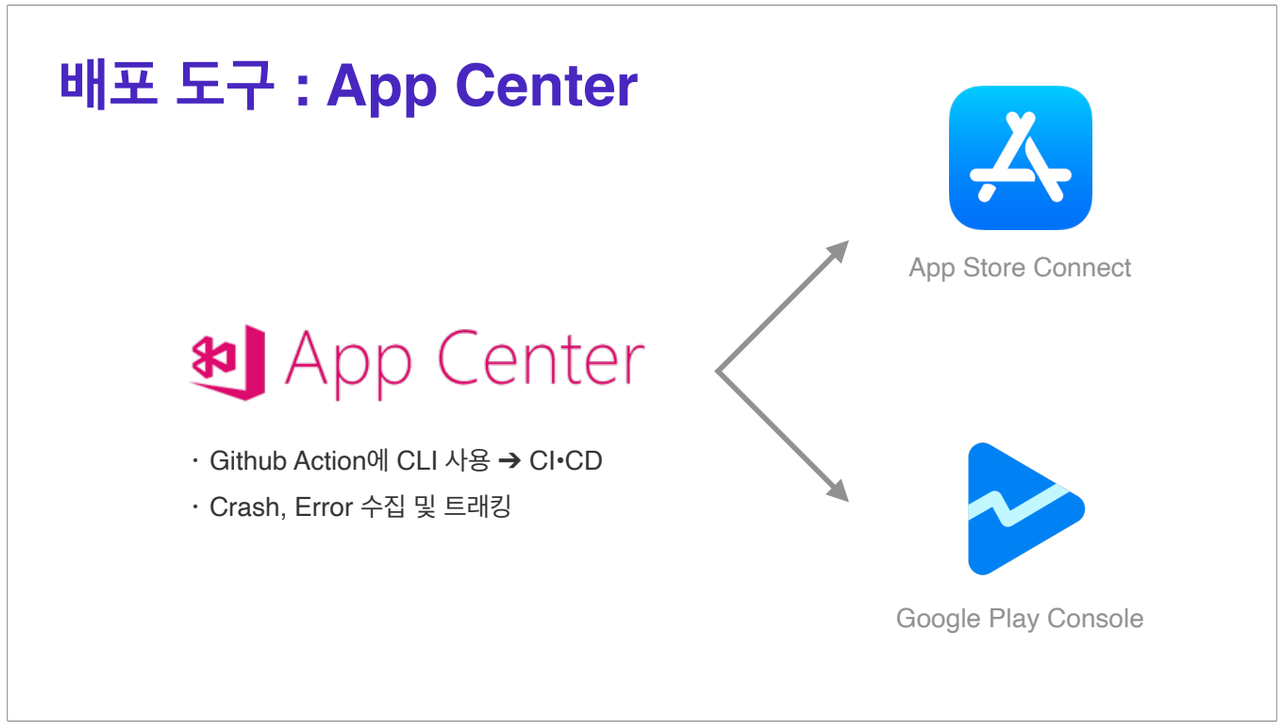
Visual Studio 2022

Visual Studio Code

- iOS앱 실행 위해 Mac과 페어링 필요

다음은 개발도구입니다 .NET 앱 개발은 IDE로는 Visual Studio, Visual Studio Code를 사용할 수 있습니다. VS for mac도 잇지만 이는 올해 8월이후 지원이 중단될 예정이라고 합니다.
iOS 앱을 개발한다면 맥이 꼭 필요한데요, 그래서 맥이 아닌 윈도우 또는 리눅스 환경에서 개발하려면 맥을 별도로 구비하고 맥과 페어링을 하셔야합니다.

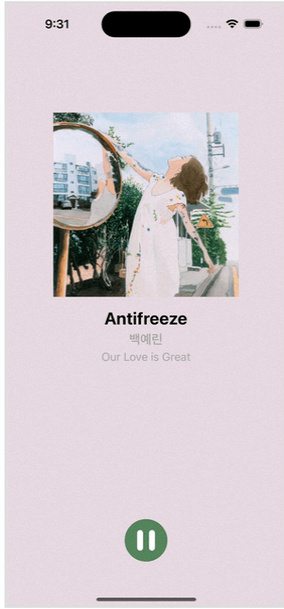
저는 맥북을 사용하고 3가지 툴을 모두 사용해본 사람으로서 후기를 공유하자면
첫번째로 VS for mac은 맥과 페어링 필요없이 빌드가 빠른 장점이 있었지만, 비쥬얼 스튜디오에 비해 코드 편집기능과, 테스팅이 매우 떨어지고 불편한 점이 많았습니다.
두번째 비주얼 스튜디오는 맥북에서 가상 머신으로 Window를 띄어 사용하는데요, 매번 맥과 페어링해야하는 번거로운 점과 빌드 속도가 느린 단점 때문에 저는 주로 테스트 코드를 작성할 때 사용합니다.
그리고 현재는 주로 Visual Studio Code를 사용해서 개발하고 있습니다. 빌드 속도도 비쥬얼 스튜디오에 비해 약 3배정도 빠르며, 비쥬얼 스튜디오의 편집기능 만큼이나 다양한 코파일럿, 깃헙등 익스텐션을 사용할 수 있어 편리하게 개발이 가능하기 때문입니다



앱 배포 준비를 마쳤다면 iOS는 앱스토어에 Android는 구글 플레이에 올려야겠죠?
앱을 각 플랫폼의 스토어에 제출하기 위해서 앱센터를 사용합니다.
앱 센터 사이트에서 수동으로도 올릴 수 있고, CLI가 있기 때문에 깃헙 액션을 사용하여 CI,CD 구축이 가능합니다
또한 크래시와 오류도 수집하고 관리 할 수 있어서, 이슈를 트래킹하는 용도로도 사용할 수 있습니다.

지금까지 개발을 위해 필요한 언어와 도구들을 알아보았습니다.
이제 예제 앱과 함께 모바일 개발 방법을 알아보도록 하겠습니다.

Demo: Music Player 🎵



오늘 만들 앱은 뮤직 플레이어 앱입니다.
이 앱은 현재 재생되고 있는 음악의 정보를 보여주고, 버튼으로 음악을 재생, 일시중지 할 수 있습니다.

Native API 사용하기

PlayMusic > Player > C# Player.iOS.cs > ...

```
#if __IOS__
using AVFoundation;
using Foundation;

namespace PlayMusic;

2 references
internal class Player
{
    3 references
    private AVAudioPlayer? _player;

    0 references
    public Player()
    {
        AVAudioSession.SharedInstance().SetCategory(AVAudioSessionCategory.Playback);
        AVAudioSession.SharedInstance().SetActive(true);
        _player = AVAudioPlayer.FromUrl(NSUrl.FromFilename("Assets/sample_music.mp3"));
    }

    0 references
    public void Play()
    {
        _player?.Play();
    }

    0 references
    public void Pause()
    {
        _player?.Pause();
    }
}
#endif
```

먼저 음악을 재생시키는 클래스를 만들어보도록하겠습니다.
iOS에서는 AVFoundation이라는 프레임워크로 음악을 재생시킵니다.

그래서 네이티브 개발과 똑같이 using문으로 프레임워크를 가져와 사용합니다.
생성자에서 음악을 재생시키는 오디오 세션을 설정하고, 파일 URI로 Player를 생성합니다.
그리고 Play, Pause메서드에서 음악을 재생/ 정지시킵니다.
Android용 Player 파일도 생성할 것이기 때문에 파일명은 Player.iOS.cs로 이름 짓습니다.

여담으로 제가 개발하고 있는 서비스는 통화서비스입니다.
그래서 입사 초기엔 네이티브에 의존적인 서비스를 크로스플랫폼에서 잘 구현할 수 있을까? 궁금하고 우려되었습니다.
하지만 지금까지 사용하려는 API가 지원되지 않는 경우는 없었고, 다행히 원활하게 개발할 수 있었습니다.
그런데 싱글코드로 두개의 플랫폼을 개발하다보니, 이 Player 클래스를 통합해주어야합니다.

ServiceContainer

- 의존성 주입(Dependency Injection) 패턴 지원

- 서비스 인스턴스 생성 및 생명주기 관리

이 때 사용할 수 있는 것이 NET의 빌트 인 ServiceContainer입니다.
서비스 컨테이너는 클래스의 의존성을 역전하기 위해 의존성 주입 패턴을 지원합니다.
서비스 컨테이너에 사용하고자 하는 서비스를 등록하면, 컨테이너는 인스턴스를 생성하고 생명주기를 관리합니다.
개발자는 생성자에서 의존성 주입을 통해 컨테이너로부터 인스턴스를 제공 받아 사용할 수 있습니다.

서비스 컨테이너를 사용하는 방법을 스텝 바이 스텝으로 코드를 보며 따라해보겠습니다.

ServiceContainer

1️⃣ Interface 정의

PlayMusic > Player > C# IPlayer.cs > IPlayer

```
namespace PlayMusic;

6 references
public interface IPlayer
{
    4 references
    public void Play();
    4 references
    public void Pause();
}
```

첫번째 Native에 대한 의존성을 가진 Player 서비스를 IPlayer라는 인터페이스로 추상화합니다

ServiceContainer

2️⃣ iOS, Android 각 파일에 Interface 구현

PlayMusic > Player > C# Player.iOS.cs > ...

```
#if __IOS__
using AVFoundation;
using Foundation;

namespace PlayMusic;

2 references
internal class Player: IPlayer
{
    3 references
    private AVAudioPlayer? _player;

    0 references
    public Player()
    {
        AVAudioSession.SharedInstance().SetCategory(AVAudioSessionCategory.Playback);
        AVAudioSession.SharedInstance().SetActive(true);
        _player = AVAudioPlayer.FromUrl(NSUrl.FromFilename("Assets/sample_music.mp3"));
    }

    2 references
    public void Play()
    {
        _player?.Play();
    }

    2 references
    public void Pause()
    {
        _player?.Pause();
    }
}
#endif
```

Player.iOS.cs

PlayMusic > Player > C# Player.Android.cs > ...

```
#if __ANDROID__
using Android.Media;

namespace PlayMusic;

2 references
internal class Player : IPlayer
{
    6 references
    private MediaPlayer _player;
    1 reference
    private string _musicURI = "Assets/sample_music.mp3";

    0 references
    public Player()
    {
        _player = new MediaPlayer();
        _player.SetDataSource(_musicURI);
        _player.Prepare();
    }
    2 references
    public void Play()
    {
        this._player.Start();
    }

    2 references
    public void Pause()
    {
        this._player.Pause();
    }
```

Player.Android.cs

두번째로, 각 플랫폼 별로 인터페이스를 구체 타입으로 구현합니다

ServiceContainer

3️⃣ 사용할 Service 등록

PlayMusic > App.cs > App > OnLaunched

```
namespace PlayMusic;

3 references
public class App : Application
{

    2 references
    protected override void OnLaunched(LaunchActivatedEventArgs args)
    {
        var builder = this.CreateBuilder(args)
            .Configure(host => host
#if DEBUG
                // Switch to Development environment when running in DEBUG
                .UseEnvironment(Environments.Development)
#endif
                .ConfigureServices((context, services) =>
                {
                    // 사용할 서비스를 등록합니다
                    services.AddSingleton<IPlayer, Player>();
                    services.AddTransient<PlayerViewModel>();

                })
            );
```

다음 서비스 컨테이너에 IPlayer와 Player 클래스를 등록합니다.

컨테이너에 등록할 때 서비스의 생명주기를 설정할 수 있는데요, 싱글톤으로 사용할 것이라 AddSingleTon을 호출해주었습니다

ServiceContainer

4️⃣ 생성자 주입으로 받은 서비스 인스턴스 사용

```
0 references
public PlayerViewModel(IPlayer player)
{
    _player = player;
}
```

마지막으로 뷰모델 생성자에서 Player를 주입 받아서 인스턴스를 사용합니다.

ViewModel

CommunityToolkit.MVVM

```
namespace PlayMusic;

3 references
public class PlayerViewModel : ObservableObject
{
    2 references
    private string _title = "Antifreeze";
    3 references
    private string _albumName = "Our Love is Great";

    0 references
    public string Title
    {
        get => _title;
        set => SetProperty(ref _title, value);
    }

    0 references
    public string AlbumName
    {
        get => _albumName;
        set => SetProperty(ref _albumName, value);
    }
```

이번엔 뷰 모델을 만들어보겠습니다.
MVVM 패턴을 좀 더 쉽게 구현하기 위해서 CommunityToolkit.Mvvm이라는 라이브러리를 사용할 수 있습니다.
이 라이브러리를 사용하면 뷰모델에 변경 사항이 생겼을 때 뷰를 자동으로 업데이트 시킬 수 있습니다.
라이브러리의 ObservableObject를 상속하고, 바인딩할 데이터의 세터에 SetProperty를 호출하면 됩니다

ViewModel

CommunityToolkit.MVVM

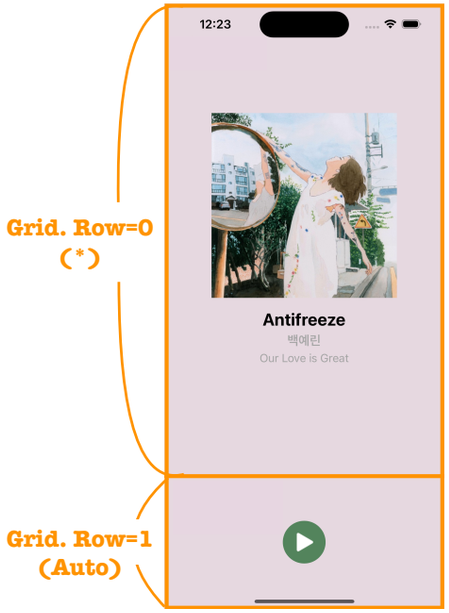
```
0 references
public ICommand PlayCommand => new RelayCommand(Play);


1 reference
private void Play()
{
    if (IsPlaying)
        _player.Pause();
    else
        _player.Play();
    IsPlaying = !IsPlaying;
}
```

그리고 컨트롤에 바인딩될 명령은 ICommand로 캡슐화하여 노출합니다.
RelayCommand는 ICommand의 구현체로 명령 실행 가능 여부를 판단하는 함수를 받을 수 있습니다.
이렇게 만들어준 뷰모델을 Page의 DataContext에 넣어주면, UI 코드에서 뷰모델에 접근해 바인딩을 할 수 있습니다.

Grid

: 레이아웃 구성

```
<Grid RowDefinitions="*, auto">
    <!-- 앨범 커버, 제목, 가수명, 앨범명 -->
    <!-- 재생 버튼 -->
</Grid>
```



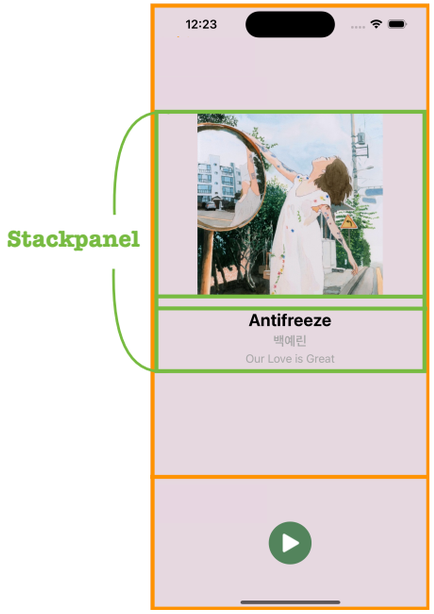
이제 UI를 구현해보겠습니다. 각 요소들이 중앙에 일렬로 간격을 두고 배치되어있는 모습입니다.
이 UI를 구현하는 방법엔 여러가지가 있겠지만 오늘은 Grid와 StackPanel을 이용해 구현해보겠습니다.

Grid는 영역을 행과 열로 구성해서 자식 요소를 배치할 수 있는 컨트롤입니다.
RowDefinition 또는 ColumnDefintion으로 몇 개의 행과 열을 사용할 건지, 길이는 어떻게 할건지 지정해줄 수 있습니다.
이 UI는 2개의 행으로 구성되고,아래 버튼이 포함된 행의 높이는 자식 요소의 사이즈를 따르는 auto로, 윗 부분은 나머지 영역을 모두 사용하겠다는 값인 “*”로 채워주도록 하겠습니다.

StackPanel

: 레이아웃 구성

```
<Grid RowDefinitions="*, auto">
    <StackPanel Grid.Row="0" Spacing="16">
        <!-- 앨범 커버 -->
        <!-- 제목, 가수명, 앨범명 -->
    </StackPanel>
    <!-- 재생 버튼 -->
</Grid>
```

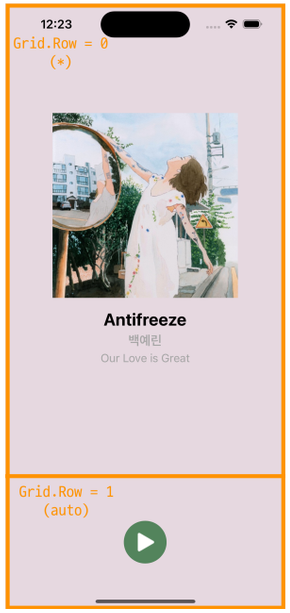


Grid가 각 행과 열의 사이즈를 지정해줄 수 있다면, StackPanel은 자식 요소의 사이즈를 반영해 수직 또는 수평적으로 일렬 배치하는 컨트롤입니다.
제목, 가수명, 앨범명의 텍스트를 하나의 스택패널로 묶고, 그 스택패널과 이미지를 또 다른 스택패널로 묶어서 각각의 행간을 다르게 지정합니다.
그리고 이렇게 묶인 StackPanel은 그리드의 1행 중앙에 배치합니다

Grid

: 레이아웃 구성

```
<Grid RowDefinitions="*, auto">
    <!-- 앨범 커버, 제목, 가수명, 앨범명 -->
    <!-- 재생 버튼 -->
</Grid>
```



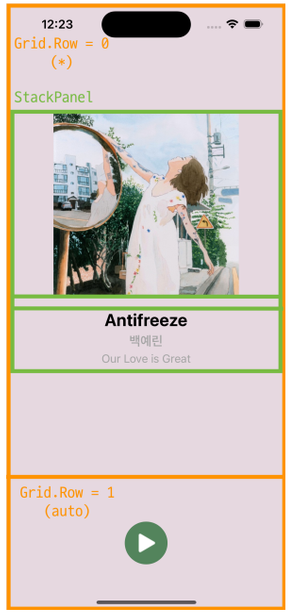
이제 UI를 구현해보겠습니다. 각 요소들이 중앙에 일렬로 간격을 두고 배치되어있는 모습입니다.
이 UI를 구현하는 방법엔 여러가지가 있겠지만 오늘은 Grid와 StackPanel을 이용해 구현해보겠습니다.

Grid는 영역을 행과 열로 구성해서 자식 요소를 배치할 수 있는 컨트롤입니다.
RowDefinition 또는 ColumnDefintion으로 몇 개의 행과 열을 사용할 건지, 길이는 어떻게 할건지 지정해줄 수 있습니다.
이 UI는 2개의 행으로 구성되고,아래 버튼이 포함된 행의 높이는 자식 요소의 사이즈를 따르는 auto로, 윗 부분은 나머지 영역을 모두 사용하겠다는 값인 “*”로 채워주도록 하겠습니다.

StackPanel

: 레이아웃 구성

```
<Grid RowDefinitions="*, auto">
    <StackPanel Grid.Row="0" Spacing="16">
        <!-- 앨범 커버 -->
        <!-- 제목, 가수명, 앨범명 -->
    </StackPanel>
    <!-- 재생 버튼 -->
</Grid>
```



Grid는 각 행과 열의 사이즈를 지정해줄 수 있다면, StackPanel은 자식 요소의 사이즈를 반영해 수직 또는 수평적으로 일렬 배치하는 컨트롤입니다.
제목, 가수명, 앨범명의 텍스트를 하나의 스택패널로 묶고, 그 스택패널과 이미지를 또 다른 스택패널로 묶어서 각각의 행간을 다르게 지정합니다.
그리고 이렇게 묶인 StackPanel은 그리드의 1행 중앙에 배치합니다

Image

: 앨범 커버 이미지 구현

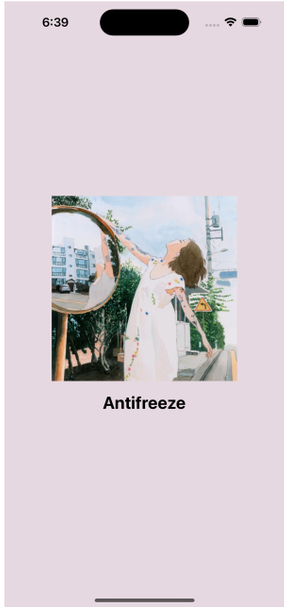
```
<Grid RowDefinitions="*, auto">
    <StackPanel Grid.Row="0" Spacing="16" VerticalAlignment="Center"
        HorizontalAlignment="Center">
        <Image Source="{Binding AlbumCover}" Width="260" Height="260"
            HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <StackPanel Spacing="4">
            <!-- 제목 -->
            <!-- 가수명 -->
            <!-- 앨범명 -->
        </StackPanel>
    </StackPanel>
    <!-- 버튼 -->
</Grid>
```

뼈대는 어떻게 표현하는지 알았으니 이제 위에서부터 내부를 채워가보겠습니다.

이미지는 jpeg, png, svg등 다양한 이미지 형식을 표현할 수 있는 컨트롤입니다.
이미지의 Source에 이미지 파일의 URI를 바인딩하고, 넓이와 높이 정렬을 설정합니다 .

TextBlock : 제목 구현

```xml
<Page.Resources>
    <Style x:Key="TextBlock.Title" TargetType="TextBlock">
        <Setter Property="FontSize" Value="24"/>
        <Setter Property="FontWeight" Value="Bold"/>
        <Setter Property="TextAlignment" Value="Center"/>
        <Setter Property="Foreground" Value="#000000"/>
    </Style>

    <Style x:Key="TextBlock.Subtitle" TargetType="TextBlock">
        <Setter Property="FontSize" Value="18"/>
        <Setter Property="FontWeight" Value="SemiBold"/>
        <Setter Property="TextAlignment"  Value="Center"/>
        <Setter Property="Foreground" Value="#A5A5A5"/>
    </Style>

    <Style x:Key="TextBlock.Body" TargetType="TextBlock">
        <Setter Property="FontSize" Value="16"/>
        <Setter Property="TextAlignment" Value="Center"/>
        <Setter Property="Foreground" Value="#A5A5A5"/>
    </Style>
</Page.Resources>
```



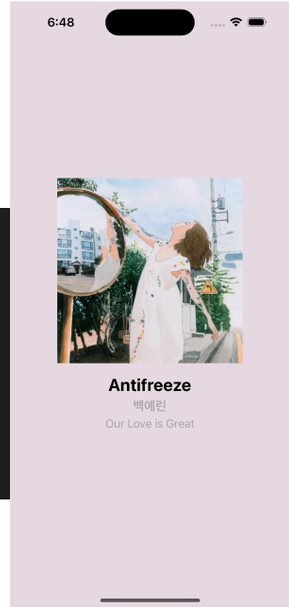
TextBlock은 텍스트를 표현하는 컨트롤입니다. 폰트 사이즈, 두께, 정렬, 색상 프로퍼티를 설정합니다.

이러한 스타일은 앱의 다른 화면에서도 재사용 될수 있는데요, Style이라는 객체를 사용하면 반복되는 컨트롤의 스타일을 묶어서 하나의 키로 관리할 수 있습니다.
스타일은 리소스 내부에 정의할 수 있으며 스타일을 적용할 타겟타입을 명시합니다.
그리고 적용할 프로퍼티와 값들을 나열하면됩니다

Style

: 재사용 가능한 스타일 정의

```
<Grid RowDefinitions="*, auto">
    <StackPanel Grid.Row="0" Spacing="16" VerticalAlignment="Center"
        HorizontalAlignment="Center">
        <Image Source="{Binding AlbumCover}" Width="260" Height="260"
            HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <StackPanel Spacing="4">
            <TextBlock Text="{Binding Title}" Style="{StaticResource TextBlock.Title}"/>
            <TextBlock Text="{Binding Artist}" Style="{StaticResource TextBlock.Subtitle}"/>
            <TextBlock Text="{Binding AlbumName}" Style="{StaticResource TextBlock.Body}"/>
        </StackPanel>
    </StackPanel>
    <!-- 버튼 -->
</Grid>
```



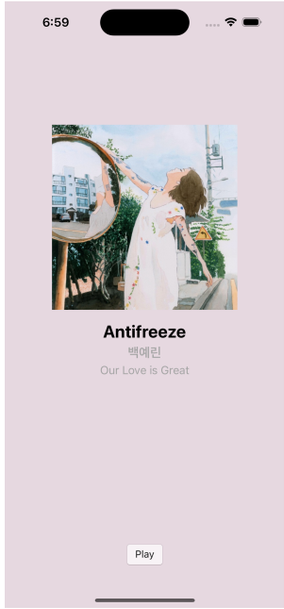
보시는 코드는 스타일을 텍스트블락의 스타일 프로퍼티에 적용한 모습입니다.

이렇게 스타일을 사용하면 디자인시스템에 표현된 컨트롤들을 한 포인트에서 관리할 수 있습니다.

Button : ContentControl

: 재생 버튼 구현

```
<Grid RowDefinitions="*, auto">
    <StackPanel Grid.Row="0" Spacing="16" VerticalAlignment="Center"
        HorizontalAlignment="Center">
        <Image Source="{Binding AlbumCover}" Width="260" Height="260"
            HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <StackPanel Spacing="4">
            <TextBlock Text="{Binding Title}" Style="{StaticResource TextBlock.Title}"/>
            <TextBlock Text="{Binding Artist}" Style="{StaticResource TextBlock.Subtitle}"/>
            <TextBlock Text="{Binding AlbumName}" Style="{StaticResource TextBlock.Body}"/>
        </StackPanel>
    </StackPanel>
    <Button Grid.Row="1" Content="Play" HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Grid>
```



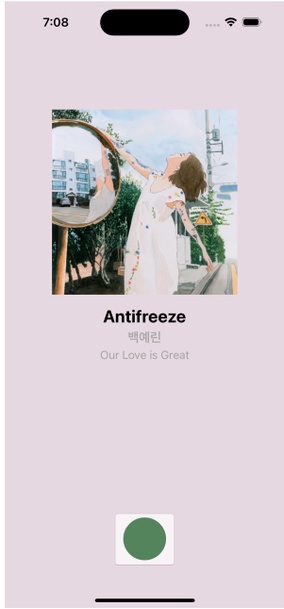
다음은 재생 버튼을 구현해보겠습니다.

Button은 ContentControl을 상속받습니다 . 하나의 컨텐트를 보여줄 수 있는데요, 컨텐트는 ContentControl 내부의 ContentPresenter라는 객체가 보여줍니다
그리고 컨텐트는 어떠한 뷰 타입이든 들어올 수 있습니다.
먼저 Play라는 텍스트를 Content에 넣어주었더니 다음과 같은 모양으로 표시됩니다.
백그라운드 컬러와 보더를 설정해주지 않는데도 위와 같이 표시되는 그 이유는 Button의 디폴트 스타일이 적용되어있기 때문입니다.

Button : ContentControl

: 재생 버튼 구현

```
<Grid RowDefinitions="*, auto">
    <StackPanel Grid.Row="0" Spacing="16" VerticalAlignment="Center"
        HorizontalAlignment="Center">
        <Image Source="{Binding AlbumCover}" Width="260" Height="260"
            HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <StackPanel Spacing="4">
            <TextBlock Text="{Binding Title}" Style="{StaticResource TextBlock.Title}"/>
            <TextBlock Text="{Binding Artist}" Style="{StaticResource TextBlock.Subtitle}"/>
            <TextBlock Text="{Binding AlbumName}" Style="{StaticResource TextBlock.Body}"/>
        </StackPanel>
    </StackPanel>
    <Button Grid.Row="1" HorizontalAlignment="Center" VerticalAlignment="Center"
        Margin="0, 60">
        <Ellipse Fill="#53845C" Width="60" Height="60"/>
    </Button>
</Grid>
```



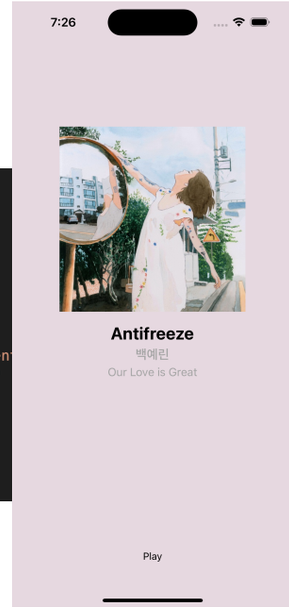
따라서 Content를 원로 넣어주면 다음과 같은 모양으로 표현되게 됩니다.

그렇다면 디폴트 스타일 대신 이 버튼 만의 비주얼을 만들고 싶을 땐 어떻게 하면될까요?

ControlTemplate

: 버튼 비주얼 재정의

```
<Style x:Key="Button.Clean" TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <ContentPresenter x:Name="ContentPresenter" Content="{TemplateBinding Content}"
                                  ContentTransitions="{TemplateBinding ContentTransitions}"
                                  ContentTemplate="{TemplateBinding ContentTemplate}"
                                  Margin="{TemplateBinding Padding}"
                                  HorizontalAlignment="{TemplateBinding HorizontalContentAlignmen
                                  VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                                  AutomationProperties.AccessibilityView="Raw"/>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```



그 땐 ControlTemplate을 사용하여 버튼의 비쥬얼을 자유롭게 만들 수 있습니다.
보시는 코드에서는 ControlTemplate의 루트 뷰를 ContentPresenter로 설정하여 Button의 Content를 보여줄 수 있게 하였습니다.

또한 TemplateBinding 이용해 컨트롤에 정의되어있는 프로퍼티의 값을 바인딩할 수도 있습니다. 예를 들어 Button의 Content를 TextBlock으로 넣고 Button의 Foreground 색상을 보라색으로 지정하면 TextBlock의 Foreground 색상도 보라로 적용되는 방식입니다.
그리고 이러한 ControlTemplate은 스타일로 정의하여 재사용 할 수 있습니다.
이렇게 만든 Button.Clean 스타일을 적용하면 다음과 같이 깨끗하게 Play만 나오는걸 확인할 수 있습니다.

DataTemplate

: 특정 데이터 타입 비주얼 정의

```
<DataTemplate x:Key="PlayButton" x:DataType="local:PlayerViewModel">
    <Button Command="{Binding PlayCommand}" Style="{StaticResource Button.Clean}">
        <Grid>
            <Ellipse Fill="#53845C" Width="60" Height="60"/>
            <Image Source="{Binding IsPlaying, Converter={StaticResource BoolToPlayPauseImageConverter}}"
                Width="50" Height="50" VerticalAlignment="Center" HorizontalAlignment="Center"/>
        </Grid>
    </Button>
</DataTemplate>
```

ControlTemplate이 컨트롤의 비주얼을 정의하는 객체였다면, 특정 데이터의 비쥬얼을 정의하는 DataTemplate도 있습니다.
ControlTemplate과 같이 루트엔 어떠한 UI든 들어올 수 있습니다.

특정 데이터 타입을 표현하는데 적합하므로 ListView 와 같이 데이터를 나열하는 UI에서 주로 사용됩니다.

데모앱에서 버튼은 IsPlaying이 true일 때는 일시정지 이미지를, IsPlaying이 False일 때는 재생 이미지를 보여줍니다.
이를 DataTemplate으로 구현해보겠습니다.
X:DataType엔 어떠한 데이터 타입을 표현할건지 타입을 적습니다.
그리고 내부엔 표현할 UI인 동그라미 위에 이미지를 올리고, 버튼의 커맨와 이미지의 소스에 뷰모델을 바인딩합니다
Image의 Source Binding옆에 Converter라는게 보이는데요. 이는 bool 값을 알맞은 URI string으로 변환해주기 위해 사용합니다.

IValueConverter : 특정 로직 수행으로 값 변환

```
using Microsoft.UI.Xaml.Data;

namespace PlayMusic;

1 reference
public class BoolToPlayPauseImageConverter : IValueConverter
{
    0 references
    public object Convert(object value, Type targetType, object parameter, string language)
    {
        bool isPlaying = (bool)value;
        return isPlaying ? "ms-appx:///PlayMusic/Assets/Icons/pause.png" : "ms-appx:///PlayMusic/Assets/Icons/play.png";
    }

    0 references
    public object ConvertBack(object value, Type targetType, object parameter, string language)
    {
        throw new NotImplementedException();
    }
}
```

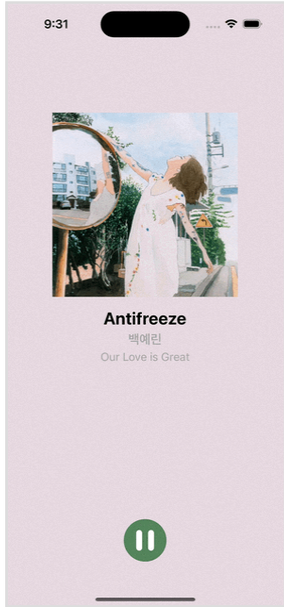
컨버터는 IValueConverter의 구현체로, 바인딩시 값에 특정 로직을 수행하여 변환한 값을 반환합니다.

Convert 와 ConvertBack 메서드를 구현하면 되는데, 바인딩된 값을 변환만 할것이므로 Convert 만 구현해주었습니다.
Convert메서드에 IsPlaying이 true일 때는 일시정지이미지를, False일 때는 재생 이미지를 반환하는 코드를 작성합니다.

IValueConverter 사용

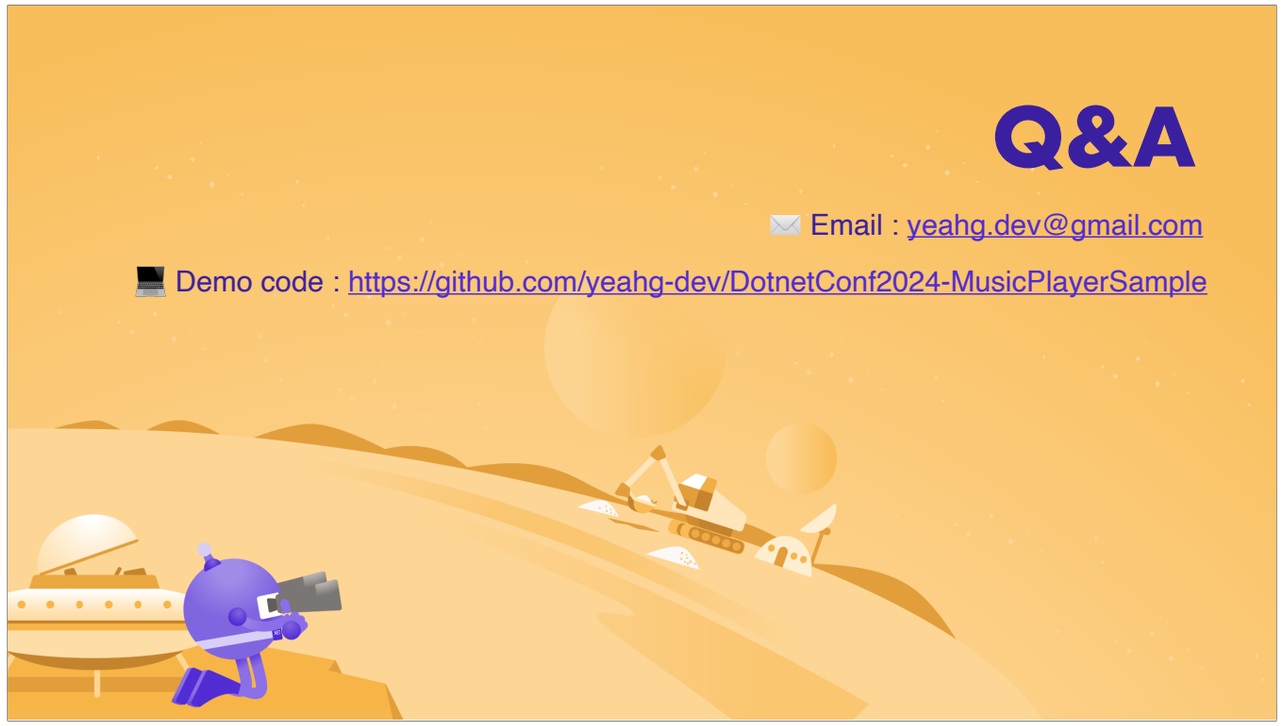
```xml
<Page.Resources>

    <local:BoolToPlayButtonImageConverter x:Key="BoolToPlayPauseImageConverter"/>

</Page.Resources>
```

```xml
<DataTemplate x:Key="PlayButton" x:DataType="local:PlayerViewModel">
    <Button Command="{Binding PlayCommand}" Style="{StaticResource Button.Clean}">
        <Grid>
            <Ellipse Fill="#53845C" Width="60" Height="60"/>
            <Image Source="{Binding IsPlaying, Converter={StaticResource BoolToPlayPauseImageConverter}}"
                Width="50" Height="50" VerticalAlignment="Center" HorizontalAlignment="Center"/>
        </Grid>
    </Button>
</DataTemplate>
```



이렇게 정의한 Converter는 Xaml에 리소스로 추가한 다음 키를 가지고 바인딩에서 사용하면 됩니다.
버튼을 누를 때매다 이미지가 변경되는 모습을 확인하실 수 있습니다.

앞서 소개드린 도구들 외에도 Xaml과 WinUI에는 반응형 UI를 구현할 수 있는 VisualStateMananger, Trigger류등 다양한 툴들이 있습니다.
.NET으로 UI 를 만들면서 느낀 점은 UI 컨트롤을 유연하게 구성하고, 덧붙여 기능도 확장할 수 있다는 점이었습니다. 그래서 원하는 어떠한 형태의 모바일 UI든 대부분은 구현이 가능할 것이라 생각하고,
또 .NET으로 모바일 앱개발을 하시는 개발자 풀이 크지 않아서 아직 정보도 적고 진입장벽이 높은 것은 사실이지만, 스타트업에서 iOS, Android 앱을 만드는데 도구적인 부족함은 없지 않았나 개인적으로 생각이 듭니다.
오늘 제가 공유드린 내용들이 .NET으로 모바일 앱을 처음 시작하시는 분들께 작은 도움이라도 되면 좋겠다는 바램과 함께 이만 오늘 발표를 마무리하겠습니다.



추후라도 궁금하신 점이 생기시면 편히 앞의 이메일 주소로 질문주시면 답변드리도록하겠습니다.

감사합니다 ☺️

Presented by aws elastic INFRAGISTICS Couchbase MVP .NET foundation

Reference

- UnoPlatfrom
- Microsoft .NET Official Site