MS MAUI 앱의 실행 순서는 크게 **프로젝트 빌드**, **플랫폼별 진입점**, **MAUI 앱 초기화**, **페이지 렌더링**의 4단계로 나눌 수 있어요.

**1. 프로젝트 빌드 🛠️**

MAUI 프로젝트를 빌드하면 .NET 런타임 환경에서 실행 가능한 형태로 변환돼요. MAUI는 **단일 프로젝트**를 사용해 여러 플랫폼(안드로이드, iOS, 윈도우, macOS)을 대상으로 빌드할 수 있어요. 빌드 과정에서 플랫폼별 네이티브 코드와 MAUI 코드가 결합돼요.

**2. 플랫폼별 진입점**

빌드된 앱을 실행하면 각 플랫폼의 네이티브 런타임이 먼저 시작돼요.

* **안드로이드**: MainActivity.cs 파일이 시작점 역할을 해요.
* **iOS/macOS**: AppDelegate.cs 파일이 시작점이에요.
* **윈도우**: App.xaml.cs 파일의 App 클래스가 시작점 역할을 해요.

이 진입점들은 모두 MauiProgram 클래스의 CreateMauiApp() 메서드를 호출해서 MAUI 앱을 초기화해요.

**3. MAUI 앱 초기화**

MauiProgram.cs 파일에 정의된 CreateMauiApp() 메서드는 앱의 구성 요소를 설정하는 중요한 역할을 해요. 여기서 다음을 포함한 앱 전반의 설정을 진행해요:

* **폰트 및 리소스 등록**: 앱에서 사용할 폰트, 이미지 등을 등록해요.
* **의존성 주입(DI)**: 서비스를 등록하고 관리해요.
* **MAUI 앱 생성**: App.xaml 파일에 정의된 App 클래스의 인스턴스를 생성해요.

**4. 페이지 렌더링**

MauiProgram에서 생성된 App 클래스는 MainPage 속성을 통해 앱의 시작 페이지를 지정해요.

* **XAML 분석**: MainPage로 지정된 XAML 파일을 분석해서 UI 요소를 생성해요.
* **네이티브 UI 컨트롤 변환**: MAUI의 추상화된 UI 컨트롤(예: Label, Button)이 각 플랫폼의 네이티브 UI 컨트롤(예: 안드로이드의 TextView, iOS의 UILabel)로 변환돼요.
* **화면 표시**: 변환된 네이티브 UI 컨트롤이 화면에 렌더링돼요. 이 과정은 **핸들러**라는 메커니즘을 통해 이루어지며, 이는 플랫폼별 구현을 분리하는 MAUI의 핵심 기능이에요.

# 폴더구조

MAUI 프로젝트를 효율적으로 관리하고 확장성을 높이기 위한 추천 폴더 구조는 주로 **기능 또는 도메인별 분리**를 기반으로 합니다. 이 구조는 MVVM(Model-View-ViewModel) 디자인 패턴을 적용하기에도 용이합니다.

**1. 기본 MAUI 프로젝트 구조**

MAUI 앱을 새로 만들면 기본적으로 다음과 같은 구조가 생성됩니다.

MyMauiApp/

├── Platforms/

│ ├── Android/

│ ├── iOS/

│ ├── MacCatalyst/

│ └── Windows/

├── Resources/

├── App.xaml

├── App.xaml.cs

├── AppShell.xaml

├── AppShell.xaml.cs

├── MauiProgram.cs

* **Platforms**: 플랫폼별 고유 코드가 들어가는 곳.
* **Resources**: 이미지, 폰트, 앱 아이콘 등 공통 리소스가 저장됩니다.
* **App.xaml**: 앱의 전역 스타일, 템플릿 등을 정의합니다.
* **AppShell.xaml**: 앱의 전체적인 셸(내비게이션, 탭바 등)을 구성합니다.
* **MauiProgram.cs**: 앱의 진입점 역할을 하며, 의존성 주입(DI) 등을 설정합니다.

**2. 추천 폴더 구조 (MVVM 기반)**

기능이 많아질수록 기본 구조만으로는 관리가 어려워집니다. 아래와 같이 **기능별**로 폴더를 나누고, 그 안에 MVVM 패턴에 따라 파일을 정리하는 것을 추천합니다.

MyMauiApp/

├── Features/

│ ├── Dashboard/

│ │ ├── Views/

│ │ │ └── DashboardPage.xaml

│ │ │ └── DashboardPage.xaml.cs

│ │ ├── ViewModels/

│ │ │ └── DashboardViewModel.cs

│ │ └── Models/

│ │ └── DashboardData.cs

│ ├── Auth/

│ │ ├── Views/

│ │ │ └── LoginPage.xaml

│ │ │ └── LoginPage.xaml.cs

│ │ ├── ViewModels/

│ │ │ └── LoginViewModel.cs

│ │ └── Models/

│ │ └── User.cs

│ └── Settings/

│ └── ...

├── Services/

│ ├── IAuthService.cs

│ ├── AuthService.cs

│ ├── IDataService.cs

│ └── DataService.cs

├── Models/

│ └── Base/

│ └── BaseEntity.cs

├── ViewModels/

│ └── Base/

│ └── BaseViewModel.cs

└── ... (기본 MAUI 파일들)

**주요 폴더 설명**

* **Features**: 앱의 주요 기능들을 모아놓는 최상위 폴더입니다. Dashboard, Auth, Settings와 같이 기능 단위로 폴더를 나눕니다.
* **Views**: UI를 정의하는 XAML 파일과 코드 비하인드(xaml.cs) 파일이 위치합니다.
* **ViewModels**: 뷰의 로직을 처리하는 뷰모델 클래스들이 위치합니다.
* **Models**: 데이터 모델 클래스들이 위치합니다.
* **Services**: API 호출, 데이터베이스 접근, 인증 등 앱의 핵심 비즈니스 로직을 담당하는 클래스들이 위치합니다. \*\*인터페이스(IAuthService)\*\*와 \*\*구현체(AuthService)\*\*로 분리하여 의존성 주입(DI)을 용이하게 만듭니다.
* **Models/Base & ViewModels/Base**: 공통적으로 사용되는 기본 클래스(예: INotifyPropertyChanged를 구현한 BaseViewModel)를 정의하여 코드 중복을 줄입니다.

이 구조를 사용하면 프로젝트의 규모가 커져도 특정 기능을 찾고 유지보수하기가 훨씬 수월해집니다.