**Avoid Heavy Operations on the Main Thread**

To avoid this, use async/await and run heavy tasks in the background. An example:

Heavy Operation

await Task.Run(() => LoadData());

**Use Compiled Bindings instead of conventional Bindings**

<Label Text="{Binding Name, Mode=OneWay}" />

<!-- Improved to: -->

<Label Text="{Binding Name, Mode=OneWay}" x:DataType="models:User" />

**Reduce the use of nested layouts**

Instead of doing something like this:

<StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Name:" />

<Label Text="{Binding Name}" />

</StackLayout>

</StackLayout>

You could use a Grid:

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<label text="name:" grid.column="0" />

<label text="{binding name}" grid.column="1" />

</Grid>

**List Virtualization**

<CollectionView ItemsSource="{Binding Users}" />

**Asynchronous Image Loading and Lazy Loading**

<ffimageloading:CachedImage Source="{Binding ImageUrl}" />

**Data Caching**

You can use libraries like MonkeyCache to handle caching in .NET MAUI applications easily.

dotnet add package MonkeyCache

**Minimizes the use of LINQ**

var result = UserList.Where(u => u.Active). ToList();

You could use a more efficient loop:

var result = new List<User>();

foreach (var user in Userslist)

{

if (user. Active)

result. Add;

}

**Performance Tracing**

Android.OS.Trace.BeginSection("Data Load");

Load logic

Android.OS.Trace.EndSection();

**Using AOT (Ahead-Of-Time) Compilation**

To enable AOT on Android:

<PropertyGroup>

<RunAOTCompilation>true</RunAOTCompilation>

</PropertyGroup>