Reduce Class Coupling with Dependency Injection in .NET MAUI Labs

Perform these labs on your own computer using Visual Studio 2022 or later to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Add IRepository Interface

Right mouse-click on the Common.Library project and add a new folder named **Interfaces**.

Right mouse-click on the **Interfaces** folder in the **Common.Library** project and add a new class named **IRepository**. Replace the entire contents of the new file with the following code.

```
using System.Collections.ObjectModel;

namespace Common.Library;

public interface IRepository<TEntity>
{
  Task<ObservableCollection<TEntity>> GetAsync();

  Task<TEntity?> GetAsync(int id);
}
```

Lab 2: Create a Data Layer Class Library

Right mouse-click on the **Solution** and add a new **Class Library** project named **AdventureWorks.DataLayer.Mock**.

Delete the **Class1.cs** file.

Right mouse-click on the new AdventureWorks.DataLayer.Mock project and add folder named **RepositoryClasses**.

Set Dependencies

Right mouse-click on the **Dependencies** folder in the **AdventureWorks.DataLayer.Mock** project and add two project references to **AdventureWorks.EntityLayer** and **Common.Library**.

Right mouse-click on the **Dependencies** folder in the **AdventureWorks.MAUI** project and a project reference to **AdventureWorks.DataLayer.Mock** project.

Add Entity Classes

Copy the **_SampleCode\EntityClasses** folder into the **AdventureWorks.EntityLayer\EntityClasses** folder.

Add Repository Classes

Copy the files from the **_SampleCode\RepositoryClasses-Mock** folder into the **AdventureWorks.DataLayer.Mock\RepositoryClasses** folder.

Try It Out

Run the application to ensure everything still works.

Lab 3: Load a User via View Model

Open the **ViewModelClasses\UserViewModel.cs** file and add a new private variable.

```
private readonly IRepository<User>? Repository;
```

Add two constructors.

```
#region Constructors
public UserViewModel(): base()
{
}

public UserViewModel(IRepository<User> repo) : base()
{
  Repository = repo;
}
#endregion
```

Modify the **GetAsync(id)** method to retrieve the data from the UserRepository class.

```
public async Task<User?> GetAsync(int id)
{
  try {
    // Get a User from a data store
    if (Repository != null) {
      CurrentEntity = await Repository.GetAsync(id);
    }
    else {
      LastErrorMessage = REPO_NOT_SET;

      // MOCK Data
      CurrentEntity = await Task.FromResult(new User {
        UserId = id,
        LoginId = "SallyJones",
        FirstName = "Sally",
        LastName = "Jones",
        Email = "Sallyj@jones.com",
        Phone = "615.987.3456",
        PhoneType = "Mobile",
        IsEnrolledIn401k = true,
        IsEnrolledInFlexTime = false,
        IsEnrolledInHealthCare = true,
        IsEnrolledInHSA = false,
        IsActive = true,
        BirthDate = Convert.ToDateTime("08-13-1989"),
        StartTime = new TimeSpan(7, 30, 0)
      });
    }

    InfoMessage = "Found the User";
    RowsAffected = 1;
  }
  catch (Exception ex) {
    RowsAffected = 0;
    PublishException(ex);
  }

  return CurrentEntity;
}
```

Lab 4: Add Repository, View Models, and Views to Dependency Injection

Open the **MauiProgram.cs** file and add some new using statements.

```
using AdventureWorks.DataLayer;
using AdventureWorks.EntityLayer;
using AdventureWorks.MAUI.Views;
using AdventureWorks.ViewModelLayer;
using Common.Library;
```

Just below the builder.UseMauiApp<App>() add the following code to inject the repository, the view model and the view into the services collection.

```
// Add services to be used in Dependency Injection
builder.Services.AddScoped<IRepository<User>,
UserRepository>();
builder.Services.AddScoped<UserViewModel>();
builder.Services.AddScoped<UserDetailView>();
```

Open the **Views\UserDetailView.xaml.cs** file and **modify** the constructor to look like the following.

```
public UserDetailView(UserViewModel viewModel)
{
  InitializeComponent();

  ViewModel = viewModel;
}
```

In the OnAppearing() override, **DELETE** the line of code that creates a new instance of the **ViewModel** property.

```
// Create a new instance of UserViewModel
ViewModel = new();
```

Try It Out

Run the application and click on the **User | Navigate to Detail** menu to see the user associated with **UserId 1** in the user repository appear.

Try other user id's such as 2, 3, 8 etc.

Lab 5: Load Phone Picker Using MVVM

Open the **ViewModelClasses\UserViewModel.cs** file and add two new private variables.

```
private readonly IRepository<PhoneType>?
_PhoneTypeRepository;
private ObservableCollection<string> _PhoneTypesList =
new();
```

Add a new public property named **PhoneTypesList**.

```
public ObservableCollection<string> PhoneTypesList
{
  get { return _PhoneTypesList; }
  set
  {
    _PhoneTypesList = value;
    RaisePropertyChanged(nameof(PhoneTypesList));
  }
}
```

Add a new method to get all phone types.

```
#region GetPhoneTypes Method
public async Task<ObservableCollection<string>>
GetPhoneTypes()
{
  if (_PhoneTypeRepository != null) {
    var list = await _PhoneTypeRepository.GetAsync();

    PhoneTypesList = new
ObservableCollection<string>(list.Select(row =>
row.TypeDescription));
  }

  return PhoneTypesList;
}
#endregion
```

Inject PhoneTypeRepository Class into Constructor

Add another constructor to accept a PhoneTypeRepository.

```
public UserViewModel(IRepository<User> repo,
IRepository<PhoneType> phoneRepo) : base()
{
  Repository = repo;
  _PhoneTypeRepository = phoneRepo;
}
```

Add Phone Types to DI

Open the **MauiProgram.cs** file and add a new service for DI to use the **PhoneTypeRepository** class.

```
builder.Services.AddScoped<IRepository<PhoneType>,
PhoneTypeRepository>();
```

Modify the User Detail View

Open the **Views\UserDetailView.xaml.cs** file and add a call to the GetPhoneTypes() method in the **OnAppearing** event procedure.

```
protected override void OnAppearing()
{
  base.OnAppearing();

  // Set the Page BindingContext
  BindingContext = ViewModel;

  // Get the Phone Types
  await ViewModel.GetPhoneTypes();

  // Retrieve a User
  await ViewModel.GetAsync(1);
}
```

Open the **App.xaml** file and **delete** the **<x:Array x:Key="phoneTypes" …>** element from the <ResourceDictionary> element.

Open the **Views\UserDetailView.xaml** file and locate the <Picker> and modify it to look like the following.

```
<Picker ItemsSource="{Binding PhoneTypesList}"
  SelectedItem="{Binding CurrentEntity.PhoneType }" />
```

Try It Out

Run the application, click on **User | Navigate to Detail** and you should still see the same list of phone types, and the picker should be positioned on the value for the user read from the repository.