Project Structure

# Project: ePicSearch

**\ePicSearch.csproj:**

﻿<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<TargetFramework>net8.0-android</TargetFramework>

<OutputType>Exe</OutputType>

<RootNamespace>ePicSearch</RootNamespace>

<UseMaui>true</UseMaui>

<SingleProject>true</SingleProject>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

<!-- Display name -->

<ApplicationTitle>ePicSearch</ApplicationTitle>

<!-- App Identifier -->

<ApplicationId>com.companyname.epicsearch</ApplicationId>

<!-- Versions -->

<ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>

<ApplicationVersion>1</ApplicationVersion>

<!-- Supported OS Platform Version for Android -->

<SupportedOSPlatformVersion>21.0</SupportedOSPlatformVersion>

<Platforms>AnyCPU</Platforms>

<!-- Optimize Android Build -->

<AndroidUseSharedRuntime>false</AndroidUseSharedRuntime>

<AndroidEnableProfiledAot>false</AndroidEnableProfiledAot>

<AndroidUseAapt2>true</AndroidUseAapt2>

</PropertyGroup>

<ItemGroup>

<!-- App Icon -->

<MauiIcon Include="Resources\AppIcon\appicon.svg" ForegroundFile="Resources\AppIcon\appiconfg.svg" />

<!-- Splash Screen -->

<MauiSplashScreen Include="Resources/Images/loading\_screen.webp" Color="#36547a" BaseSize="2048,2048" />

<!-- Images -->

<MauiImage Include="Resources\Images\\*.webp" />

<!-- Custom Fonts -->

<MauiFont Include="Resources\Fonts\\*" />

<!-- Raw Assets -->

<MauiAsset Include="Resources\Raw\\*\*" LogicalName="%(RecursiveDir)%(Filename)%(Extension)" />

</ItemGroup>

<ItemGroup>

<None Remove="Platforms\Android\Resources\values\styles.xml" />

</ItemGroup>

<ItemGroup>

<PackageReference Include="CommunityToolkit.Maui" Version="9.1.0" />

<PackageReference Include="Microsoft.Maui.Controls" Version="8.0.91" />

<PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="8.0.0" />

<PackageReference Include="Microsoft.Maui.Essentials" Version="8.0.91" />

<PackageReference Include="Newtonsoft.Json" Version="13.0.3" />

<PackageReference Include="Serilog" Version="4.0.2" />

<PackageReference Include="Serilog.Extensions.Logging.File" Version="3.0.0" />

<PackageReference Include="Serilog.Sinks.File" Version="6.0.0" />

</ItemGroup>

<ItemGroup>

<ProjectReference Include="..\ePicSearch.Core\ePicSearch.Infrastructure.csproj" />

</ItemGroup>

<ItemGroup>

<Compile Update="Views\GamePage.xaml.cs">

<DependentUpon>GamePage.xaml</DependentUpon>

</Compile>

<Compile Update="Views\SplashCreen.xaml.cs">

<DependentUpon>SplashCreen.xaml</DependentUpon>

</Compile>

</ItemGroup>

<ItemGroup>

<MauiXaml Update="Views\MyAdventuresPage.xaml">

<Generator>MSBuild:Compile</Generator>

</MauiXaml>

<MauiXaml Update="Views\NewAdventurePage.xaml">

<Generator>MSBuild:Compile</Generator>

</MauiXaml>

<MauiXaml Update="Views\GamePage.xaml">

<Generator>MSBuild:Compile</Generator>

</MauiXaml>

<MauiXaml Update="Views\SplashCreen.xaml">

<Generator>MSBuild:Compile</Generator>

</MauiXaml>

<MauiXaml Update="Views\SettingsPage.xaml">

<Generator>MSBuild:Compile</Generator>

</MauiXaml>

</ItemGroup>

</Project>

**\App.xaml:**

﻿<?xml version = "1.0" encoding = "UTF-8" ?>

<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:ePicSearch"

x:Class="ePicSearch.App">

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="Resources/Styles/Colors.xaml" />

<ResourceDictionary Source="Resources/Styles/Styles.xaml" />

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

</Application>

**\App.xaml.cs:**

﻿using ePicSearch.Infrastructure.Services;

namespace ePicSearch

{

public partial class App : Application

{

private readonly DataStorageService \_dataStorageService;

public App(AppShell appShell, DataStorageService dataStorageService)

{

InitializeComponent();

MainPage = appShell;

\_dataStorageService = dataStorageService;

}

// Sync cache when the app goes to sleep (background or closing)

protected override void OnSleep()

{

\_dataStorageService.SyncCacheToFile();

base.OnSleep();

}

}

}

**\AppShell.xaml:**

<Shell

x:Class="ePicSearch.AppShell"

xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:views="clr-namespace:ePicSearch.Views"

Shell.FlyoutBehavior="Disabled"

Title="ePicSearch">

<ShellContent

ContentTemplate="{DataTemplate views:MainPage}"

Route="MainPage" />

</Shell>

**\AppShell.xaml.cs:**

﻿using ePicSearch.Services;

using ePicSearch.Views;

namespace ePicSearch

{

public partial class AppShell : Shell

{

public AppShell(MainPage mainPage)

{

InitializeComponent();

Items.Add(new ShellContent

{

Title = "Home",

Content = mainPage

});

}

}

}

**\MauiProgram.cs:**

﻿using Microsoft.Extensions.Logging;

using ePicSearch.Infrastructure.Services;

using ePicSearch.Views;

using ePicSearch.Infrastructure.Entities.Interfaces;

using ePicSearch.Services;

using Serilog;

using CommunityToolkit.Maui;

namespace ePicSearch

{

public static class MauiProgram

{

public static MauiApp CreateMauiApp()

{

var builder = MauiApp.CreateBuilder();

var logFilePath = Path.Combine(FileSystem.AppDataDirectory, "logs.txt");

Log.Logger = new LoggerConfiguration()

.MinimumLevel.Debug() // Set minimum log level

.Enrich.FromLogContext()

.WriteTo.File(

logFilePath,

outputTemplate: "{Timestamp:yyyy-MM-dd HH:mm:ss.fff } [{Level:u3}] [{SourceContext}] {Message:lj}{NewLine}{Exception}",

rollOnFileSizeLimit: true, // Roll over when file size exceeds the limit

fileSizeLimitBytes: 10 \* 1024 \* 1024, // 10 MB file size limit

retainedFileCountLimit: 1) // Keep only the last log file

.CreateLogger();

builder

.UseMauiApp<App>()

.UseMauiCommunityToolkit()

.ConfigureFonts(fonts =>

{

fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");

fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");

});

// Register services with the app data directory

builder.Services

.AddSingleton<IFileSystemService, FileSystemService>()

.AddSingleton<DataStorageService>()

.AddSingleton<PhotoStorageService>()

.AddSingleton<CodeGenerator>()

.AddSingleton<AdventureManager>()

.AddSingleton<MainPage>()

.AddSingleton<AppShell>();

builder.Logging.AddSerilog();

#if DEBUG

builder.Logging.AddDebug();

#endif

return builder.Build();

}

}

}

**Entities\AppFileInfo.cs:**

﻿using ePicSearch.Infrastructure.Entities.Interfaces;

namespace ePicSearch.Entities

{

public class AppFileResult(FileResult fileResult) : IFileResult

{

private readonly FileResult \_fileResult = fileResult;

public string FullPath => \_fileResult.FullPath;

public string FileName => \_fileResult.FileName;

}

}

**Services\FileSystemService.cs:**

﻿using Microsoft.Maui.Storage;

using ePicSearch.Infrastructure.Entities.Interfaces;

namespace ePicSearch.Services

{

public class FileSystemService : IFileSystemService

{

public string GetAppDataDirectory() => FileSystem.AppDataDirectory;

public bool FileExists(string path) => File.Exists(path);

public string ReadAllText(string path) => File.ReadAllText(path);

public void WriteAllText(string path, string content) => File.WriteAllText(path, content);

}

}

**Views\GamePage.xaml:**

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="ePicSearch.Views.GamePage">

<StackLayout Padding="10" Spacing="20">

<Label Text="{Binding AdventureName}" FontSize="24" HorizontalOptions="Center" TextColor="Black" />

<CollectionView x:Name="PhotoCollectionView"

ItemsSource="{Binding Photos}"

HorizontalScrollBarVisibility="Never"

HorizontalOptions="FillAndExpand"

VerticalOptions="CenterAndExpand">

<CollectionView.ItemsLayout>

<!-- Set to Horizontal List -->

<LinearItemsLayout Orientation="Horizontal" />

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<Grid Padding="5">

<!-- Base Layer: Actual Photo -->

<Image Source="{Binding Photo.FilePath}"

Aspect="AspectFill"

HeightRequest="150"

WidthRequest="150">

<Image.GestureRecognizers>

<TapGestureRecognizer Command="{Binding BindingContext.ShowPhotoCommand, Source={x:Reference PhotoCollectionView}}"

CommandParameter="{Binding .}" />

</Image.GestureRecognizers>

</Image>

<!-- Overlay Layer: Question Mark -->

<Image Source="question\_mark1.webp"

IsVisible="{Binding IsLocked}"

Aspect="AspectFit"

HeightRequest="150"

WidthRequest="150"

Opacity="0.6" />

</Grid>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

<!-- Full-Screen Photo Modal (Initially Hidden) -->

<ContentView x:Name="PhotoModal" IsVisible="False" BackgroundColor="#000000AA">

<Grid VerticalOptions="Center" HorizontalOptions="Center">

<!-- Full-Screen Photo or Question Mark -->

<Image x:Name="FullScreenPhoto" Aspect="AspectFit" HeightRequest="300" WidthRequest="300" />

<!-- Code Entry Overlay when Locked -->

<StackLayout x:Name="CodeEntryOverlay" IsVisible="False" BackgroundColor="#FFFFFFDD" Padding="20" VerticalOptions="End">

<Entry x:Name="FullScreenCodeEntry" Placeholder="Enter unlock code" WidthRequest="200"/>

<Button Text="Unlock" Command="{Binding UnlockPhotoCommand}"

CommandParameter="{Binding Text, Source={x:Reference FullScreenCodeEntry}}" />

</StackLayout>

</Grid>

</ContentView>

</StackLayout>

</ContentPage>

**Views\GamePage.xaml.cs:**

using ePicSearch.Entities;

using ePicSearch.Infrastructure.Entities;

using ePicSearch.Infrastructure.Services;

using Microsoft.Extensions.Logging;

using System.Collections.ObjectModel;

using System.Windows.Input;

namespace ePicSearch.Views

{

public partial class GamePage : ContentPage

{

public ICommand ShowPhotoCommand { get; }

public ICommand UnlockPhotoCommand { get; }

public ObservableCollection<PhotoDisplayInfo> Photos { get; private set; }

public string AdventureName { get; private set; }

private readonly ILogger<MainPage> \_logger;

private readonly AdventureManager \_photoManager;

private PhotoDisplayInfo? \_selectedPhoto;

public GamePage(string adventureName, ILogger<MainPage> logger, AdventureManager photoManager)

{

InitializeComponent();

\_logger = logger;

\_logger.LogInformation($"GamePage for adventure: {adventureName} initialized");

AdventureName = adventureName;

Photos = new ObservableCollection<PhotoDisplayInfo>();

ShowPhotoCommand = new Command<PhotoDisplayInfo>(ShowPhoto);

UnlockPhotoCommand = new Command<string>(UnlockPhoto);

\_photoManager = photoManager;

\_selectedPhoto = null;

LoadAdventurePhotos(adventureName);

}

private void LoadAdventurePhotos(string adventureName)

{

var photos = \_photoManager.GetPhotosForAdventure(adventureName);

\_logger.LogInformation($"Photos for adventure: {adventureName} were loaded : {photos}");

if (photos != null && photos.Count > 0)

{

// Order photos by SerialNumber descending (latest first)

var orderedPhotos = photos.OrderByDescending(p => p.SerialNumber).ToList();

Photos = new ObservableCollection<PhotoDisplayInfo>(

orderedPhotos.Select((photo, index) => new PhotoDisplayInfo(photo, index, orderedPhotos.Count)));

RefreshPhotoView();

\_logger.LogInformation($"All photos for adventure: {adventureName} were loaded into Observable Collection");

}

else

{

\_logger.LogWarning($"No photos were found for adventure: {adventureName}");

DisplayAlert("No Photos", "No photos found for this adventure.", "OK");

}

}

private void ShowPhoto(PhotoDisplayInfo photoInfo)

{

\_logger.LogInformation($"A photo {photoInfo.ToString} was pressed");

\_selectedPhoto = photoInfo;

// Set the selected photo to be displayed in the full-screen modal

if (photoInfo.Photo.IsLocked)

{

// Show the locked overlay image

FullScreenPhoto.Source = "question\_mark\_1.webp";

CodeEntryOverlay.IsVisible = true;

}

else

{

FullScreenPhoto.Source = photoInfo.Photo.FilePath;

CodeEntryOverlay.IsVisible = false;

}

// Display the modal

PhotoModal.IsVisible = true;

}

private void UnlockPhoto(string code)

{

\_logger.LogInformation($"A code {code} was proposed to unlock a photo");

if (\_selectedPhoto == null)

{

\_logger.LogWarning("No photos were selected");

return;

}

if (\_selectedPhoto.Photo.Code == code)

{

\_selectedPhoto.Photo.IsLocked = false;

RefreshPhotoView();

if (\_photoManager.UpdatePhotoState(\_selectedPhoto.Photo))

{

\_logger.LogInformation($"The photo with {code} was successfully updated in memory");

}

else

{

\_logger.LogError($"{\_selectedPhoto} failed to be updated in memory");

DisplayAlert("Save Error", "Failed to save the changes. Please try again.", "OK");

// Revert the change in the UI to keep things consistent

\_selectedPhoto.Photo.IsLocked = true;

RefreshPhotoView();

}

}

else

{

\_logger.LogInformation($"Thecode for {\_selectedPhoto} was wrong");

}

}

private void RefreshPhotoView()

{

PhotoCollectionView.ItemsSource = null;

PhotoCollectionView.ItemsSource = Photos;

}

}

}

**Views\MainPage.xaml:**

﻿<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:blur="clr-namespace:ePicSearch.Behaviors"

x:Class="ePicSearch.Views.MainPage">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="150" />

<!-- Button rows-->

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<!-- Low Button rows-->

<RowDefinition Height="120" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Image x:Name="BackgroundImage"

Source="mainpage\_background.webp"

Aspect="AspectFill"

Grid.RowSpan="5"

Grid.ColumnSpan="3">

<Image.Behaviors>

<blur:BlurBehavior Radius="0" x:Name="BlurEffect"/>

</Image.Behaviors>

</Image>

<!-- My Adventures Button -->

<ImageButton Source="start\_adventure\_button.webp"

Clicked="OnMyAdventuresClicked"

x:Name="MyAdventuresButton"

WidthRequest="180"

Aspect="AspectFit"

Grid.Row="1"

Grid.Column="1"

HorizontalOptions="Center"

VerticalOptions="Start"

Opacity="0"/>

<!-- Create Adventure Button -->

<ImageButton Source="create\_adventure\_button.webp"

Clicked="OnCreateNewAdventureClicked"

x:Name="CreateAdventureButton"

WidthRequest="200"

Aspect="AspectFit"

Grid.Row="2"

Grid.Column="1"

HorizontalOptions="Center"

VerticalOptions="Start"

Opacity="0"/>

<!-- Exit Button in Row 0, Column 0 -->

<ImageButton Source="quit\_button.webp"

Clicked="OnQuitClicked"

x:Name="QuitButton"

WidthRequest="60"

HeightRequest="60"

Grid.Row="4"

Grid.Column="0"

HorizontalOptions="Center"

VerticalOptions="Center"

Opacity="0"/>

<!-- Settings Button in Row 0, Column 2 -->

<ImageButton Source="settings\_button.webp"

Clicked="OnSettingsClicked"

x:Name="SettingsButton"

WidthRequest="60"

HeightRequest="60"

Grid.Row="4"

Grid.Column="2"

HorizontalOptions="Center"

VerticalOptions="Center"

Opacity="0"/>

</Grid>

</ContentPage>

**Views\MainPage.xaml.cs:**

﻿using ePicSearch.Behaviors;

using ePicSearch.Infrastructure.Services;

using Microsoft.Extensions.Logging;

namespace ePicSearch.Views

{

public partial class MainPage : ContentPage

{

private readonly AdventureManager \_photoManager;

private readonly ILogger<MainPage> \_logger;

private bool \_isBlurred = false;

public MainPage(AdventureManager photoManager, ILogger<MainPage> logger)

{

InitializeComponent();

\_photoManager = photoManager;

\_logger = logger;

Appearing += MainPage\_Appearing;

\_logger.LogInformation("MainPage initialized.");

}

private async void MainPage\_Appearing(object? sender, EventArgs e)

{

if (!\_isBlurred)

{

// Display the background for 1 second

await Task.Delay(1000);

var zoomTask = BackgroundImage.ScaleTo(1.1, 2000, Easing.CubicInOut);

var blurBehavior = BackgroundImage.Behaviors.OfType<BlurBehavior>().FirstOrDefault();

var blurTask = blurBehavior?.AnimateBlurEffect(10, 100);

var fadeTask = BackgroundImage.FadeTo(0.7, 2000, Easing.CubicInOut);

var buttonsAppearTask = ButtonsAppear(MyAdventuresButton, CreateAdventureButton, SettingsButton, QuitButton);

await Task.WhenAll(zoomTask, blurTask!, fadeTask, buttonsAppearTask);

\_isBlurred = true;

}

}

private async Task ButtonsAppear(params ImageButton[] buttons)

{

var animationTasks = new List<Task>();

foreach (var button in buttons)

{

var buttonTask = Task.WhenAll(

button.FadeTo(1, 1000),

button.TranslateTo(0, -10, 1000, Easing.CubicInOut)

);

animationTasks.Add(buttonTask);

}

await Task.WhenAll(animationTasks);

}

private async void OnSettingsClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new SettingsPage());

}

private async void OnMyAdventuresClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new MyAdventuresPage(\_photoManager, \_logger));

}

private async void OnCreateNewAdventureClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new NewAdventurePage(\_photoManager));

}

private void OnQuitClicked(object sender, EventArgs e)

{

Application.Current?.Quit();

}

}

}

**Views\MyAdventuresPage.xaml:**

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="ePicSearch.Views.MyAdventuresPage">

<StackLayout Padding="10" VerticalOptions="FillAndExpand">

<Label Text="Your Adventures" FontSize="20" HorizontalOptions="Center" />

<!-- no adventures -->

<Grid x:Name="NoAdventuresGrid" IsVisible="False" VerticalOptions="CenterAndExpand">

<Label x:Name="NoAdventuresLabel"

Text="No adventures found"

FontSize="18"

TextColor="Gray"

HorizontalOptions="Center"

VerticalOptions="Center" />

</Grid>

<!-- CollectionView for Adventures -->

<CollectionView x:Name="AdventuresList"

IsVisible="True">

<CollectionView.ItemTemplate>

<DataTemplate>

<Frame BackgroundColor="White" Padding="10" Margin="10" HasShadow="True">

<StackLayout Orientation="Vertical" Spacing="10" HorizontalOptions="CenterAndExpand">

<!-- Adventure title -->

<Label Text="{Binding .}" HorizontalOptions="Center" VerticalOptions="Center" FontSize="18" TextColor="Black"/>

<!-- Buttons -->

<StackLayout Orientation="Horizontal" Spacing="20" HorizontalOptions="Center">

<Button Text="Play" Clicked="OnPlayAdventureClicked" CommandParameter="{Binding .}" />

<Button Text="Delete" Clicked="OnDeleteAdventureClicked" CommandParameter="{Binding .}" />

</StackLayout>

</StackLayout>

</Frame>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

<!-- "Delete All" button, only shown when adventures exist -->

<Button x:Name="DeleteAllButton" Text="Delete All" Clicked="OnDeleteAllAdventuresClicked" HorizontalOptions="Center" IsVisible="True" />

</StackLayout>

</ContentPage>

**Views\MyAdventuresPage.xaml.cs:**

using ePicSearch.Infrastructure.Services;

using Microsoft.Extensions.Logging;

namespace ePicSearch.Views

{

public partial class MyAdventuresPage : ContentPage

{

private readonly AdventureManager \_adventureManager;

private readonly ILogger<MainPage> \_logger;

// Use Dependency Injection to provide PhotoManager instance

public MyAdventuresPage(AdventureManager photoManager, ILogger<MainPage> logger)

{

InitializeComponent();

\_adventureManager = photoManager;

\_logger = logger;

LoadAdventures();

}

// Load adventures from JSON to display in the UI

private void LoadAdventures()

{

\_logger.LogInformation($"Attempting to load all adventures");

var adventures = \_adventureManager.GetAllAdventureNames();

if (adventures.Count > 0)

{

AdventuresList.ItemsSource = adventures;

AdventuresList.IsVisible = true;

DeleteAllButton.IsVisible = true;

NoAdventuresGrid.IsVisible = false;

}

else

{

AdventuresList.IsVisible = false;

DeleteAllButton.IsVisible = false;

NoAdventuresGrid.IsVisible = true;

}

\_logger.LogInformation($"Done loading adventures");

}

private async void OnPlayAdventureClicked(object sender, EventArgs e)

{

if (sender is Button button && button.CommandParameter is string adventureName)

{

\_logger.LogInformation($"Attempting to play adventure {adventureName}");

await Navigation.PushAsync(new GamePage(adventureName, \_logger, \_adventureManager));

}

}

private async void OnDeleteAdventureClicked(object sender, EventArgs e)

{

if (sender is Button button && button.CommandParameter is string adventureName)

{

bool confirm = await DisplayAlert($"Confirm delete {adventureName}", null, "Yes", "No");

if (confirm)

{

try

{

\_logger.LogInformation($"Attempting to delete adventure: {adventureName}");

// Delete all photos and adventure folder

bool photosDeleted = await \_adventureManager.DeleteAdventureAsync(adventureName);

if (photosDeleted)

{

\_logger.LogInformation($"Successfully deleted adventure: {adventureName}");

}

else

{

\_logger.LogWarning($"Failed to delete photos for adventure: {adventureName}");

await DisplayAlert("Error", "Could not delete the photos for the adventure.", "OK");

}

LoadAdventures();

}

catch (Exception ex)

{

\_logger.LogError(ex, $"Error deleting adventure: {adventureName}");

await DisplayAlert("Error", $"Failed to delete adventure {adventureName}: {ex.Message}", "OK");

}

}

}

}

private async void OnDeleteAllAdventuresClicked(object sender, EventArgs e)

{

bool confirm = await DisplayAlert("Confirm Delete All", "Are you sure you want to delete all adventures?", "Yes", "No");

if (confirm)

{

try

{

\_logger.LogInformation("Attempting to delete all adventures.");

var allAdventureNames = \_adventureManager.GetAllAdventureNames();

bool allDeleted = true;

foreach (var adventureName in allAdventureNames)

{

bool success = await \_adventureManager.DeleteAdventureAsync(adventureName);

if (!success)

{

\_logger.LogWarning($"Failed to delete adventure: {adventureName}");

allDeleted = false;

}

}

if (allDeleted)

{

\_logger.LogInformation("All adventures successfully deleted.");

await DisplayAlert("Success", "All adventures have been deleted.", "OK");

}

else

{

\_logger.LogWarning("Some adventures could not be deleted.");

await DisplayAlert("Warning", "Some adventures could not be deleted properly.", "OK");

}

LoadAdventures(); // Refresh the UI

}

catch (Exception ex)

{

\_logger.LogError(ex, "Error deleting all adventures.");

await DisplayAlert("Error", $"An error occurred: {ex.Message}", "OK");

}

}

}

}

}

**Views\NewAdventurePage.xaml:**

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="ePicSearch.Views.NewAdventurePage">

<StackLayout Padding="10" Spacing="20">

<Label Text="Create a New Adventure" FontSize="20" HorizontalOptions="Center" />

<Entry x:Name="AdventureNameEntry" Placeholder="Enter adventure name" />

<Button Text="Start Creating" Clicked="OnStartCreatingClicked" />

<Label x:Name="ErrorMessage" Text="" TextColor="Red" IsVisible="False" />

</StackLayout>

</ContentPage>

**Views\NewAdventurePage.xaml.cs:**

using ePicSearch.Infrastructure.Services;

using ePicSearch.Infrastructure.Entities.Interfaces;

using ePicSearch.Entities;

using ePicSearch.Infrastructure.Entities;

namespace ePicSearch.Views

{

public partial class NewAdventurePage : ContentPage

{

private readonly AdventureManager \_adventureManager;

public NewAdventurePage(AdventureManager adventureManager)

{

InitializeComponent();

\_adventureManager = adventureManager;

}

private async void OnStartCreatingClicked(object sender, EventArgs e)

{

var adventureName = await GetValidAdventureNameAsync();

if (adventureName == null)

{

return; // Validation failed, exit the method.

}

bool keepTakingPhotos = true;

while (keepTakingPhotos)

{

if (MediaPicker.IsCaptureSupported)

{

try

{

FileResult? photo = await MediaPicker.CapturePhotoAsync();

if (photo == null)

{

await DisplayAlert("No Photo", null, "OK");

return;

}

IFileResult appFileResult = new AppFileResult(photo);

var photoInfo = await \_adventureManager.CapturePhoto(appFileResult, adventureName);

if (photoInfo == null)

{

await HandleError("Failed to save the photo", sender, e);

return;

}

await DisplayPhotoSavedMessage(photoInfo);

keepTakingPhotos = await DisplayAlert("Another clue?", null, "Yes", "No");

}

catch (Exception ex)

{

await HandleError(ex.Message, sender, e);

return;

}

}

else

{

await HandleError("Camera is not supported.", sender, e);

return;

}

}

await DisplayAlert($"Adventure {adventureName} Saved", null, "OK");

await Navigation.PopAsync();

}

private async Task HandleError(string errorMessage, object sender, EventArgs e)

{

bool continueProcess = await DisplayAlert("Error", $"{errorMessage} Your progress has been saved. Do you want to continue?", "Yes", "No");

if (continueProcess)

{

OnStartCreatingClicked(sender, e);

}

else

{

await Navigation.PopAsync();

}

}

private async Task<string?> GetValidAdventureNameAsync()

{

string adventureName = AdventureNameEntry.Text;

if (string.IsNullOrEmpty(adventureName))

{

ErrorMessage.Text = "Please enter a name for your adventure.";

ErrorMessage.IsVisible = true;

return null;

}

var existingAdventures = \_adventureManager.GetAllAdventureNames();

if (existingAdventures.Contains(adventureName, StringComparer.OrdinalIgnoreCase))

{

await DisplayAlert("Oops!", "This adventure name already exists. Please choose a different name.", "OK");

return null;

}

return adventureName;

}

private async Task DisplayPhotoSavedMessage(PhotoInfo photoInfo)

{

if (photoInfo.SerialNumber == 1)

{

await DisplayAlert("Treasure Photo Saved!", $"Code: {photoInfo.Code}, go hide it!", "OK");

}

else

{

await DisplayAlert("Clue Photo Saved!", $"Code: {photoInfo.Code}, go hide it!", "OK");

}

}

}

}

**Views\SettingsPage.xaml:**

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="ePicSearch.Views.SettingsPage">

<StackLayout VerticalOptions="CenterAndExpand" HorizontalOptions="CenterAndExpand" Spacing="20">

<!--Label Text="Settings Page" /-->

<Button Text="Clean Logs" Clicked="OnCleanLogsClicked" />

</StackLayout>

</ContentPage>

**Views\SettingsPage.xaml.cs:**

namespace ePicSearch.Views

{

public partial class SettingsPage : ContentPage

{

public SettingsPage()

{

InitializeComponent();

}

private void OnCleanLogsClicked(object sender, EventArgs e)

{

var logFilePath = Path.Combine(FileSystem.Current.AppDataDirectory, "logs.txt");

try

{

// Open the file and overwrite it with an empty string

File.WriteAllText(logFilePath, string.Empty);

DisplayAlert("Success", "Log file has been cleared.", "OK");

}

catch (Exception ex)

{

DisplayAlert("Error", $"Failed to clear log: {ex.Message}", "OK");

}

}

}

}

**Behaviors\BlurBehavior.Android.cs:**

﻿using Android.Graphics;

using Android.Widget;

namespace ePicSearch.Behaviors

{

public partial class BlurBehavior : PlatformBehavior<Image, ImageView>

{

ImageView? imageView;

protected override void OnAttachedTo(Image bindable, ImageView platformView)

{

imageView = platformView;

SetRendererEffect(platformView, Radius);

}

protected override void OnDetachedFrom(Image bindable, ImageView platformView)

{

SetRendererEffect(platformView, 0);

}

void SetRendererEffect(ImageView imageView, float radius)

{

if (OperatingSystem.IsAndroidVersionAtLeast(31))

{

var renderEffect = radius > 0 ? GetEffect(radius) : null;

imageView.SetRenderEffect(renderEffect);

}

}

static RenderEffect? GetEffect(float radius)

{

return OperatingSystem.IsAndroidVersionAtLeast(31) ?

RenderEffect.CreateBlurEffect(radius, radius, Shader.TileMode.Decal!) :

null;

}

}

}

**Behaviors\BlurBehavior.cs:**

﻿using Microsoft.Maui.Controls;

namespace ePicSearch.Behaviors

{

public partial class BlurBehavior

{

public static readonly BindableProperty RadiusProperty = BindableProperty.Create(

nameof(Radius), typeof(float), typeof(BlurBehavior), 5f, propertyChanged: OnRadiusChanged);

public float Radius

{

get => (float)GetValue(RadiusProperty);

set => SetValue(RadiusProperty, value);

}

static void OnRadiusChanged(BindableObject bindable, object oldValue, object newValue)

{

var behavior = (BlurBehavior)bindable;

if (behavior.imageView is null)

{

return;

}

behavior.SetRendererEffect(behavior.imageView, Convert.ToSingle(newValue));

}

public async Task AnimateBlurEffect(int maxRadius = 10, int delay = 100)

{

for (int i = 0; i <= maxRadius; i++)

{

Radius = i;

await Task.Delay(delay); // Adjust the delay if needed

}

}

}

}

## Files in Resources\Images:

create\_adventure\_button.png

create\_adventure\_button.webp

loading\_screen.webp

mainpage\_background.webp

question\_mark\_1.webp

question\_mark\_2.webp

quit\_button.webp

settings\_button.webp

start\_adventure\_button.webp

# Project: ePicSearch.Core

**\ePicSearch.Infrastructure.csproj:**

﻿<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<TargetFramework>net8.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Newtonsoft.Json" Version="13.0.3" />

<PackageReference Include="Serilog" Version="4.0.2" />

<PackageReference Include="Serilog.Extensions.Logging.File" Version="3.0.0" />

<PackageReference Include="Serilog.Sinks.File" Version="6.0.0" />

</ItemGroup>

</Project>

**Entities\PhotoDisplayInfo.cs:**

﻿using ePicSearch.Infrastructure.Entities;

namespace ePicSearch.Entities

{

public class PhotoDisplayInfo

{

public PhotoInfo Photo { get; set; }

public string DisplaySerialNumber { get; set; } = "";

public bool ShouldShowCode { get; set; } = false;

public PhotoDisplayInfo(PhotoInfo photo, int index, int totalPhotos)

{

Photo = photo;

Photo.IsLocked = (index != totalPhotos - 1);

DisplaySerialNumber = (index == totalPhotos - 1) ? "Treasure!" : (index + 1).ToString();

ShouldShowCode = (index != 0);

}

}

}

**Entities\PhotoInfo.cs:**

﻿using System.Xml.Linq;

namespace ePicSearch.Infrastructure.Entities

{

public class PhotoInfo

{

public string FilePath { get; set; } = "";

public string Name { get; set; } = "";

public string Code { get; set; } = "";

public int SerialNumber { get; set; }

public string AdventureName { get; set; } = "";

public bool IsLocked { get; set; }

public override string ToString()

{

return $"PhotoInfo [FilePath: {FilePath}, Name: {Name}, Code: {Code}, SerialNumber: {SerialNumber}, AdventureName: {AdventureName}, IsLocked: {IsLocked}]";

}

}

}

**Entities\Interfaces\IFileResult.cs:**

﻿namespace ePicSearch.Infrastructure.Entities.Interfaces

{

public interface IFileResult

{

string FullPath { get; }

string FileName { get; }

}

}

**Entities\Interfaces\IFileSystemService.cs:**

﻿namespace ePicSearch.Infrastructure.Entities.Interfaces

{

public interface IFileSystemService

{

string GetAppDataDirectory();

bool FileExists(string path);

string ReadAllText(string path);

void WriteAllText(string path, string content);

}

}

**Services\AdventureManager.cs:**

﻿using ePicSearch.Infrastructure.Entities;

using ePicSearch.Infrastructure.Entities.Interfaces;

using Microsoft.Extensions.Logging;

namespace ePicSearch.Infrastructure.Services

{

public class AdventureManager(PhotoStorageService photoStorageService,

CodeGenerator codeGenerator,

DataStorageService dataStorageService,

ILogger<AdventureManager> logger)

{

private readonly PhotoStorageService \_photoStorageService = photoStorageService;

private readonly CodeGenerator \_codeGenerator = codeGenerator;

private readonly DataStorageService \_dataStorageService = dataStorageService;

private readonly ILogger<AdventureManager> \_logger = logger;

public async Task<PhotoInfo?> CapturePhoto(IFileResult photo, string adventureName)

{

try

{

string photoCode = \_codeGenerator.GenerateCode();

int serialNumber = GetNextSerialNumber(adventureName);

var photoInfo = new PhotoInfo

{

FilePath = photo.FullPath,

Name = $"{photoCode}\_{serialNumber}",

Code = photoCode,

AdventureName = adventureName,

SerialNumber = serialNumber,

IsLocked = true

};

// Save photo

var newFilePath = await \_photoStorageService.SavePhotoAsync(photo, photoInfo);

if (string.IsNullOrEmpty(newFilePath))

{

\_logger.LogWarning($"Failed to save photo for adventure: {adventureName}");

return null;

}

photoInfo.FilePath = newFilePath;

\_dataStorageService.AddPhoto(photoInfo);

\_logger.LogInformation($"Photo captured and saved successfully for adventure: {adventureName}");

return photoInfo;

}

catch (Exception ex)

{

\_logger.LogError(ex, $"Error capturing photo for adventure: {adventureName}");

return null;

}

}

public bool UpdatePhotoState(PhotoInfo updatedPhoto)

{

try

{

\_dataStorageService.UpdatePhoto(updatedPhoto);

\_logger.LogInformation($"Photo state updated for photo: {updatedPhoto.FilePath}");

return true;

}

catch (Exception ex)

{

\_logger.LogError(ex, $"Error updating photo state for photo: {updatedPhoto.FilePath}");

return false;

}

}

public List<string> GetAllAdventureNames() => \_dataStorageService.GetAllAdventureNames();

public List<PhotoInfo> GetPhotosForAdventure(string adventureName) => \_dataStorageService.GetPhotosForAdventure(adventureName);

public async Task<bool> DeleteAdventureAsync(string adventureName)

{

try

{

// Delete photos from storage

var deleteResult = await \_photoStorageService.DeleteAllPhotosForAdventureAsync(adventureName);

if (deleteResult == PhotoStorageService.DeleteFolderResult.Failure)

{

\_logger.LogWarning($"Failed to delete photos for adventure: {adventureName}");

return false;

}

// Remove adventure from data storage

\_dataStorageService.RemoveAdventure(adventureName);

\_logger.LogInformation($"Adventure deleted successfully: {adventureName}");

\_dataStorageService.SyncCacheToFile();

return true;

}

catch (Exception ex)

{

\_logger.LogError(ex, $"Error deleting adventure: {adventureName}");

return false;

}

}

private int GetNextSerialNumber(string adventureName)

{

var photos = \_dataStorageService.GetPhotosForAdventure(adventureName);

return photos.Select(p => p.SerialNumber).DefaultIfEmpty(0).Max() + 1;

}

}

}

**Services\CodeGenerator.cs:**

﻿namespace ePicSearch.Infrastructure.Services

{

public class CodeGenerator

{

private Random \_random = new Random();

public string GenerateCode()

{

return \_random.Next(1000, 9999).ToString();

}

}

}

**Services\DataStorageService.cs:**

﻿using ePicSearch.Infrastructure.Entities;

using ePicSearch.Infrastructure.Entities.Interfaces;

using Microsoft.Extensions.Logging;

using Newtonsoft.Json;

namespace ePicSearch.Infrastructure.Services

{

public class DataStorageService

{

private readonly IFileSystemService \_fileSystemService;

private readonly ILogger<DataStorageService> \_logger;

private readonly string \_jsonFilePath;

private readonly object \_cacheLock = new object();

private List<PhotoInfo> \_cache;

private bool \_isCacheDirty = false;

public DataStorageService(IFileSystemService fileSystemService, ILogger<DataStorageService> logger)

{

\_fileSystemService = fileSystemService;

\_jsonFilePath = Path.Combine(\_fileSystemService.GetAppDataDirectory(), "adventures.json");

\_logger = logger;

\_logger.LogInformation($"Initialized with file path: {\_jsonFilePath}");

\_cache = LoadAdventuresFromFile();

}

public List<PhotoInfo> LoadAdventuresFromFile()

{

lock (\_cacheLock)

{

\_logger.LogInformation($"Loading adventures from JSON.");

if (!\_fileSystemService.FileExists(\_jsonFilePath))

{

\_logger.LogWarning($"JSON file not found. Returning empty adventure list.");

return new List<PhotoInfo>();

}

try

{

var json = \_fileSystemService.ReadAllText(\_jsonFilePath);

var adventures = JsonConvert.DeserializeObject<List<PhotoInfo>>(json) ?? new List<PhotoInfo>();

\_logger.LogInformation($"Loaded {adventures.Count} adventures from JSON.");

return adventures;

}

catch (Exception ex)

{

\_logger.LogError(ex, $"Error loading adventures from JSON.");

return new List<PhotoInfo>();

}

}

}

public void RemoveAdventure(string adventureName)

{

lock (\_cacheLock)

{

\_cache.RemoveAll(p => p.AdventureName == adventureName);

\_isCacheDirty = true;

}

}

public List<PhotoInfo> GetPhotosForAdventure(string adventureName)

{

lock (\_cacheLock)

{

return \_cache.Where(p => p.AdventureName == adventureName).ToList();

}

}

public void AddPhoto(PhotoInfo photoInfo)

{

lock (\_cacheLock)

{

\_cache.Add(photoInfo);

\_isCacheDirty = true;

}

}

public void UpdatePhoto(PhotoInfo updatedPhoto)

{

lock (\_cacheLock)

{

var existingPhoto = \_cache.FirstOrDefault(p => p.FilePath == updatedPhoto.FilePath);

if (existingPhoto != null)

{

existingPhoto.IsLocked = updatedPhoto.IsLocked;

\_isCacheDirty = true;

}

}

}

public List<string> GetAllAdventureNames()

{

lock (\_cacheLock)

{

return \_cache.Select(p => p.AdventureName).Distinct().ToList();

}

}

public void SyncCacheToFile()

{

lock (\_cacheLock)

{

if (!\_isCacheDirty)

{

\_logger.LogInformation("Cache is clean. No need to sync with JSON.");

return;

}

try

{

\_logger.LogInformation("Syncing cache with JSON file.");

\_logger.LogInformation($"cache written : \n {string.Join(",\n", \_cache)}");

var json = JsonConvert.SerializeObject(\_cache, Formatting.Indented);

\_fileSystemService.WriteAllText(\_jsonFilePath, json);

\_isCacheDirty = false;

\_logger.LogInformation("Cache successfully synced to JSON.");

}

catch (Exception ex)

{

\_logger.LogError(ex, "Error syncing cache to JSON.");

}

}

}

}

}

**Services\PhotoStorageService .cs:**

﻿using ePicSearch.Infrastructure.Entities;

using ePicSearch.Infrastructure.Entities.Interfaces;

using Microsoft.Extensions.Logging;

namespace ePicSearch.Infrastructure.Services

{

public class PhotoStorageService(IFileSystemService fileSystemService, ILogger<PhotoStorageService> logger)

{

private readonly string \_appDataDirectory = fileSystemService.GetAppDataDirectory();

private readonly ILogger<PhotoStorageService> \_logger = logger;

public async Task<string> SavePhotoAsync(IFileResult photo, PhotoInfo photoInfo)

{

try

{

string adventureFolderPath = Path.Combine(\_appDataDirectory, photoInfo.AdventureName);

if (!Directory.Exists(adventureFolderPath))

{

Directory.CreateDirectory(adventureFolderPath);

}

string fileExtension = Path.GetExtension(photo.FileName);

string newFilePath = Path.Combine(adventureFolderPath, $"{photoInfo.Name}{fileExtension}");

// Copy the photo from the original path to the adventure filder

using var sourceStream = File.OpenRead(photo.FullPath);

using var destinationStream = File.Create(newFilePath);

await sourceStream.CopyToAsync(destinationStream);

DeletePhoto(photo.FullPath);

\_logger.LogInformation($"Photo saved to {newFilePath}");

return newFilePath;

}

catch (Exception ex)

{

\_logger.LogError(ex, $"Failed to save photo: {ex.Message}");

return string.Empty;

}

}

public async Task<DeleteFolderResult> DeleteAllPhotosForAdventureAsync(string adventureName)

{

string adventureFolderPath = Path.Combine(\_appDataDirectory, adventureName);

try

{

if (Directory.Exists(adventureFolderPath))

{

\_logger.LogInformation($"Deleting folder for adventure: {adventureName}");

Directory.Delete(adventureFolderPath, true);

return DeleteFolderResult.Success;

}

else

{

\_logger.LogWarning($"Adventure folder not found: {adventureName}");

return DeleteFolderResult.NotFound;

}

}

catch (Exception ex)

{

\_logger.LogError(ex, $"Error deleting folder for adventure: {adventureName}");

return DeleteFolderResult.Failure;

}

}

private void DeletePhoto(string filePath)

{

try

{

if (File.Exists(filePath))

{

File.Delete(filePath);

\_logger.LogInformation($"Deleted original photo: {filePath}");

}

}

catch (Exception ex)

{

\_logger.LogError(ex, $"Error deleting original photo: {filePath}");

}

}

public enum DeleteFolderResult

{

Success,

NotFound,

Failure

}

}

}

# Total files: 38