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" Color="#512BD4" />

<!-- Splash Screen -->

<MauiSplashScreen Include="Resources\Splash\splash.svg" Color="#512BD4" BaseSize="128,128" />

<!-- Images -->

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

<MauiImage Update="Resources\Images\dotnet\_bot.png" Resize="True" BaseSize="300,185" />

<!-- Custom Fonts -->

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

<!-- Raw Assets -->

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

</ItemGroup>

<ItemGroup>

<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>

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

<Generator>MSBuild:Compile</Generator>

</MauiXaml>

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

<Generator>MSBuild:Compile</Generator>

</MauiXaml>

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

<Generator>MSBuild:Compile</Generator>

</MauiXaml>

<MauiXaml Update="Views\ViewAdventurePage.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>

**\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;

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>()

.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<JsonStorageService>()

.AddSingleton<PhotoStorageService>()

.AddSingleton<CodeGenerator>()

.AddSingleton<PhotoManager>()

.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\MainPage.xaml:**

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

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

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

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

<!-- Button for Settings -->

<Button x:Name="SettingsBtn" Text="Settings" Clicked="OnSettingsClicked" />

<!-- Button for My Adventures -->

<Button x:Name="MyAdventuresBtn" Text="My Adventures" Clicked="OnMyAdventuresClicked" />

<!-- Button for New Adventure -->

<Button x:Name="CreateNewAdventureBtn" Text="Create New Adventure" Clicked="OnCreateNewAdventureClicked" />

<!-- Button for Quit App -->

<Button x:Name="QuitBtn" Text="Quit" Clicked="OnQuitClicked" />

</StackLayout>

</ContentPage>

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

﻿using ePicSearch.Infrastructure.Services;

using Microsoft.Extensions.Logging;

namespace ePicSearch.Views

{

public partial class MainPage : ContentPage

{

private readonly PhotoManager \_photoManager;

private readonly ILogger<MainPage> \_logger;

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

{

InitializeComponent();

\_photoManager = photoManager;

\_logger = logger;

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

}

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>

<!-- ListView for Adventures -->

<ListView x:Name="AdventuresList"

HasUnevenRows="True"

IsVisible="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<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="View" Clicked="OnViewAdventureClicked" CommandParameter="{Binding .}" />

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

</StackLayout>

</StackLayout>

</Frame>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

<!-- "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 PhotoManager \_photoManager;

private readonly ILogger<MainPage> \_logger;

// Use Dependency Injection to provide PhotoManager instance

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

{

InitializeComponent();

\_photoManager = photoManager;

\_logger = logger;

LoadAdventures();

}

// Load adventures from JSON to display in the UI

private void LoadAdventures()

{

var adventures = \_photoManager.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;

}

}

private async void OnViewAdventureClicked(object sender, EventArgs e)

{

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

{

await Navigation.PushAsync(new ViewAdventurePage(adventureName, \_photoManager));

}

}

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 = \_photoManager.DeleteAdventure(adventureName);

if (photosDeleted)

{

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

LoadAdventures();

}

else

{

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

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

}

}

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 = \_photoManager.GetAllAdventureNames();

bool allDeleted = true;

foreach (var adventureName in allAdventureNames)

{

bool success = \_photoManager.DeleteAdventure(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 PhotoManager \_photoManager;

public NewAdventurePage(PhotoManager photoManager)

{

InitializeComponent();

\_photoManager = photoManager;

}

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 \_photoManager.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 = \_photoManager.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");

}

}

}

}

**Views\ViewAdventurePage.xaml:**

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

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

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

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

<StackLayout Padding="10">

<Label x:Name="AdventureNameLabel" FontSize="24" HorizontalOptions="Center" TextColor="Black" />

<ListView x:Name="PhotoListView" HasUnevenRows="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid Padding="10">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="150" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<!-- Photo -->

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

HeightRequest="150"

WidthRequest="150"

Aspect="AspectFill"

Grid.Column="0" />

<!-- Photo Data -->

<StackLayout Grid.Column="1"

VerticalOptions="Center"

HorizontalOptions="CenterAndExpand"

Spacing="5">

<Label Text="Code:" FontSize="14" TextColor="Gray" IsVisible="{Binding ShouldShowCode}" />

<Label Text="{Binding Photo.Code}" FontSize="16" TextColor="Black" IsVisible="{Binding ShouldShowCode}" />

<Label Text="{Binding DisplaySerialNumber}" FontSize="16" TextColor="Black" />

</StackLayout>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

**Views\ViewAdventurePage.xaml.cs:**

using ePicSearch.Entities;

using ePicSearch.Infrastructure.Services;

namespace ePicSearch.Views

{

public partial class ViewAdventurePage : ContentPage

{

private readonly PhotoManager \_photoManager;

public ViewAdventurePage(string adventureName, PhotoManager photoManager)

{

InitializeComponent();

\_photoManager = photoManager;

AdventureNameLabel.Text = adventureName;

LoadPhotos(adventureName);

}

private void LoadPhotos(string adventureName)

{

var photos = \_photoManager.GetPhotosForAdventure(adventureName);

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

{

// Order photos by SerialNumber descending (latest first)

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

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

PhotoListView.ItemsSource = displayPhotos;

}

else

{

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

}

}

}

}

# 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;

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

ShouldShowCode = (index != 0);

}

}

}

**Entities\PhotoInfo.cs:**

﻿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; } = "";

}

}

**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\CodeGenerator.cs:**

﻿namespace ePicSearch.Infrastructure.Services

{

public class CodeGenerator

{

private Random \_random = new Random();

public string GenerateCode()

{

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

}

}

}

**Services\JsonStorageService.cs:**

﻿using ePicSearch.Infrastructure.Entities;

using ePicSearch.Infrastructure.Entities.Interfaces;

using Microsoft.Extensions.Logging;

using Newtonsoft.Json;

namespace ePicSearch.Infrastructure.Services

{

public class JsonStorageService

{

private readonly IFileSystemService \_fileSystemService;

private readonly ILogger<JsonStorageService> \_logger;

private readonly string \_jsonFilePath;

private List<PhotoInfo> \_cache; // Cache to minimize I/O

private bool \_isCacheDirty = false; // Track changes

public JsonStorageService(IFileSystemService fileSystemService, ILogger<JsonStorageService> 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()

{

\_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 List<PhotoInfo> LoadAdventuresFromJson()

{

\_logger.LogInformation("Fetching adventures from cache.");

return \_cache;

}

public bool SaveAdventuresToJson(List<PhotoInfo> adventures)

{

\_logger.LogInformation($"Updating cache with {adventures.Count} adventures.");

\_cache = new List<PhotoInfo>(adventures);

\_isCacheDirty = true;

\_logger.LogInformation("Cache updated");

return true;

}

public List<string> GetAllAdventureNames()

{

\_logger.LogInformation($"Fetching all adventure names from cache.");

return \_cache

.Select(p => p.AdventureName)

.Distinct()

.ToList();

}

public List<PhotoInfo> GetPhotosForAdventure(string adventureName)

{

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

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

}

public void SyncCacheToFile()

{

if (!\_isCacheDirty)

{

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

return;

}

try

{

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

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

\_fileSystemService.WriteAllText(\_jsonFilePath, json);

\_isCacheDirty = false; // Reset dirty flag

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

}

catch (Exception ex)

{

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

}

}

}

}

**Services\PhotoManager.cs:**

﻿using ePicSearch.Infrastructure.Entities;

using ePicSearch.Infrastructure.Entities.Interfaces;

using Microsoft.Extensions.Logging;

namespace ePicSearch.Infrastructure.Services

{

public class PhotoManager(PhotoStorageService photoStorageService,

CodeGenerator codeGenerator,

JsonStorageService jsonStorageService,

ILogger<PhotoManager> logger)

{

private readonly PhotoStorageService \_photoStorageService = photoStorageService;

private readonly CodeGenerator \_codeGenerator = codeGenerator;

private readonly JsonStorageService \_jsonStorageService = jsonStorageService;

private readonly ILogger<PhotoManager> \_logger = logger;

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

{

string photoCode = \_codeGenerator.GenerateCode();

int serialNumber = GetNextAvailableSerialNumberForAdventure(adventureName);

var photoInfo = new PhotoInfo

{

FilePath = photo.FullPath,

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

Code = photoCode,

AdventureName = adventureName,

SerialNumber = serialNumber

};

// 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;

if (!AddPhotoToAdventure(photoInfo))

{

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

return null; // Return null if JSON update fails

}

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

return photoInfo;

}

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

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

public bool DeleteAdventure(string adventureName)

{

var photos = GetPhotosForAdventure(adventureName);

bool folderDeleted = \_photoStorageService.DeleteAdventureFolder(adventureName);

if (folderDeleted)

{

if (RemovePhotosFromJson(photos))

{

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

return true;

}

else

{

\_logger.LogWarning($"Failed to update JSON after deleting adventure: {adventureName}");

}

}

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

return false;

}

private bool AddPhotoToAdventure(PhotoInfo photoInfo)

{

var adventures = \_jsonStorageService.LoadAdventuresFromJson();

adventures.Add(photoInfo);

return \_jsonStorageService.SaveAdventuresToJson(adventures);

}

private bool RemovePhotosFromJson(List<PhotoInfo> photos)

{

var allPhotos = \_jsonStorageService.LoadAdventuresFromJson();

allPhotos.RemoveAll(p => photos.Any(photo => photo.FilePath == p.FilePath));

if(\_jsonStorageService.SaveAdventuresToJson(allPhotos))

{

\_logger.LogInformation("Adventure photos removed from JSON and synced.");

return true;

}

\_logger.LogError("Failed to update JSON after removing adventure photos.");

return false;

}

private int GetNextAvailableSerialNumberForAdventure(string adventureName)

{

var adventures = \_jsonStorageService.LoadAdventuresFromJson();

return adventures

.Where(p => p.AdventureName == adventureName)

.Select(p => p.SerialNumber)

.DefaultIfEmpty(0)

.Max() + 1;

}

}

}

**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);

}

// Append the correct file extension

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);

if (!DeletePhoto(photo.FullPath))

{

\_logger.LogWarning($"Failed to delete original photo: {photo.FullPath}");

}

return newFilePath;

}

catch (Exception ex)

{

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

return string.Empty;

}

}

public bool DeletePhoto(string filePath)

{

if (File.Exists(filePath))

{

try

{

if (File.Exists(filePath))

{

File.Delete(filePath);

\_logger.LogInformation($"Photo deleted: {filePath}");

return true;

}

else

{

\_logger.LogWarning($"Photo not found: {filePath}");

return false;

}

}

catch (Exception ex)

{

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

return false;

}

}

return false;

}

public bool DeleteAdventureFolder(string adventureName)

{

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

try

{

if (Directory.Exists(adventureFolderPath))

{

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

Directory.Delete(adventureFolderPath, true); // Recursively delete contents

}

else

{

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

}

return true;

}

catch (Exception ex)

{

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

return false;

}

}

public string GetPhotoPath(string fileName, string adventureName)

{

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

string fullPath = Path.Combine(adventureFolderPath, fileName);

if (File.Exists(fullPath))

{

return fullPath;

}

else

{

throw new FileNotFoundException($"Photo file not found: {fullPath}");

}

}

}

}

# Total files: 26