FaceAppWithGPT2

FaceAppWithGPT2

Dependencies

- DlibDotNet v19.21.0.20220724
- Emgu.CV v4.9.0.5494
- Emgu.CV.runtime.windows v4.9.0.5494
- Xabe.FFmpeg v5.2.6

/FaceAppWithGPT2/Program.cs

```
namespace FaceAppWithGPT2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

ImageProcessingLibrary

Dependencies

- DlibDotNet v19.21.0.20220724
- Emgu.CV v4.9.0.5494
- Emgu.CV.runtime.windows v4.9.0.5494
- Xabe.FFmpeg v5.2.6

/ImageProcessingLibrary/Helpers/DirectoryHelper.cs

```
using System;
using System.Collections.Generic;
using System.IO;

namespace ImageProcessingLibrary.Helpers
{
    public static class DirectoryHelper
    {
        /// <summary>
        /// Validates if the given directory path exists. If it doesn't exist, throws a Dire
        /// </summary>
        /// <param name="directoryPath">The path of the directory to validate.</param>
        public static void ValidateDirectory(string directoryPath)
```

```
        {

            if (directoryPath == null)
                throw new ArgumentNullException(nameof(directoryPath), "Directory path canno
            if (string.IsNullOrWhiteSpace(directoryPath))
                throw new ArgumentException("Directory path cannot be empty.", nameof(direct

            if (!Directory.Exists(directoryPath))
                throw new DirectoryNotFoundException($"Directory '{directoryPath}' not found
        }

        /// <summary>
        /// Gets all image files (JPG, PNG) from the specified directory.
        /// </summary>
        /// <param name="directoryPath">The path of the directory to search for image files
        /// <returns>A list of file paths for the images found in the directory.</returns>
        public static List<string> GetImageFiles(string directoryPath)
        {
            ValidateDirectory(directoryPath);

            // Define allowed image extensions
            string[] allowedExtensions = { ".jpg", ".jpeg", ".png" };

            // Get all files with allowed extensions
            var imageFiles = new List<string>();
            foreach (var file in Directory.GetFiles(directoryPath))
            {
                if (Array.Exists(allowedExtensions, ext => ext.Equals(Path.GetExtension(file
                {
                    imageFiles.Add(file);
                }
            }

            return imageFiles;
        }
    }
}
```

/ImageProcessingLibrary/Interfaces/IImageResizer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ImageProcessingLibrary.Interfaces
{
    internal interface IImageResizer
    {
        void ResizeImage(string inputPath, string outputPath, int width, int height);
    }
}
```

/ImageProcessingLibrary/PictureSizeAdaptation/ImageResizer.cs

```
using Emgu.CV;
using ImageProcessingLibrary.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageProcessingLibrary.PictureSizeAdaptation
{
    public class ImageResizer : IImageResizer
    {
        public void ResizeImage(string inputPath, string outputPath, int width, int height)
        {
            // Validate input paths
            if (!File.Exists(inputPath))
            {
                throw new FileNotFoundException($"Input file not found: {inputPath}");
            }

            // Load the image using Emgu.CV
            using (Mat image = CvInvoke.Imread(inputPath))
            {
                // Resize the image while maintaining the aspect ratio
                Mat resizedImage = new Mat();
                CvInvoke.Resize(image, resizedImage, new System.Drawing.Size(width, height),

                // Save the resized image to the output path
                CvInvoke.Imwrite(outputPath, resizedImage);
            }
        }
    }
}
```

FaceMorphingLibrary

Dependencies

- DlibDotNet v19.21.0.20220724
- Emgu.CV v4.9.0.5494
- Emgu.CV.runtime.windows v4.9.0.5494
- Xabe.FFmpeg v5.2.6

/FaceMorphingLibrary/Class1.cs

```
namespace FaceMorphingLibrary
{
    public class Class1
    {

    }
}
```

VideoGenerationLibrary

Dependencies

- DlibDotNet v19.21.0.20220724
- Emgu.CV v4.9.0.5494
- Emgu.CV.runtime.windows v4.9.0.5494
- Xabe.FFmpeg v5.2.6

/VideoGenerationLibrary/Class1.cs

```
namespace VideoGenerationLibrary
{
    public class Class1
    {

    }
}
```

ImageProcessingLibrary.Tests

Dependencies

- coverlet.collector v6.0.0
- Emgu.CV.runtime.windows v4.9.0.5494
- Microsoft.NET.Test.Sdk v17.8.0
- NUnit v3.14.0
- NUnit.Analyzers v3.9.0
- NUnit3TestAdapter v4.5.0

/ImageProcessingLibrary.Tests/DirectoryHelperTests.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ImageProcessingLibrary.Helpers;
using NUnit.Framework;

namespace ImageProcessingLibrary.Tests
{
    [TestFixture]
    public class DirectoryHelperTests
    {
        [Test]
        public void ValidateDirectory_ShouldThrowArgumentNullException_WhenPathIsNull()
        {
            // Act & Assert
            Assert.Throws<ArgumentNullException>(() => DirectoryHelper.ValidateDirectory(nul
        }

        [Test]
        public void ValidateDirectory_ShouldThrowArgumentException_WhenPathIsEmpty()
        {
            // Act & Assert
            Assert.Throws<ArgumentException>(() => DirectoryHelper.ValidateDirectory(""));
        }

        [Test]
        public void ValidateDirectory_ShouldThrowDirectoryNotFoundException_WhenDirectoryDoe
        {
            // Arrange
            string nonExistentDirectory = "C:\\NonExistentDirectory";

            // Act & Assert
            Assert.Throws<DirectoryNotFoundException>(() => DirectoryHelper.ValidateDirecto
        }

        [Test]
        public void ValidateDirectory_ShouldNotThrowException_WhenDirectoryExists()
        {
            // Arrange
            string existingDirectory = Path.GetTempPath();
```

```
    // Act & Assert
    Assert.DoesNotThrow(() => DirectoryHelper.ValidateDirectory(existingDirectory));
}

[Test]
public void GetImageFiles_ShouldReturnEmptyList_WhenNoImagesArePresent()
{
    // Arrange
    string tempDirectory = Path.Combine(Path.GetTempPath(), "EmptyDirectory");
    Directory.CreateDirectory(tempDirectory);

    try
    {
        // Act
        List<string> imageFiles = DirectoryHelper.GetImageFiles(tempDirectory);

        // Assert
        Assert.AreEqual(0, imageFiles.Count);
    }
    finally
    {
        // Cleanup
        Directory.Delete(tempDirectory);
    }
}

[Test]
public void GetImageFiles_ShouldReturnImageFiles_WhenImagesArePresent()
{
    // Arrange
    string tempDirectory = Path.Combine(Path.GetTempPath(), "ImageDirectory");
    Directory.CreateDirectory(tempDirectory);

    string imagePath1 = Path.Combine(tempDirectory, "image1.jpg");
    string imagePath2 = Path.Combine(tempDirectory, "image2.png");
    File.Create(imagePath1).Dispose();
    File.Create(imagePath2).Dispose();

    try
    {
        // Act
        List<string> imageFiles = DirectoryHelper.GetImageFiles(tempDirectory);

        // Assert
        Assert.AreEqual(2, imageFiles.Count);
        Assert.Contains(imagePath1, imageFiles);
```

```
                Assert.Contains(imagePath2, imageFiles);
            }
            finally
            {
                // Cleanup
                Directory.Delete(tempDirectory, true);
            }
        }
    }
}
```

/ImageProcessingLibrary.Tests/ImageResizerTests.cs

```
using System;
using System.Collections.Generic;
using System.Drawing.Imaging;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ImageProcessingLibrary.PictureSizeAdaptation;
using NUnit.Framework;

namespace ImageProcessingLibrary.Tests
{
    [TestFixture]
    public class ImageResizerTests
    {
        [Test]
        public void ResizeImage_ShouldThrowFileNotFoundException_WhenInputFileDoesNotExist()
        {
            // Arrange
            var imageResizer = new ImageResizer();
            string nonExistentFilePath = "C:\\NonExistentFile.jpg";
            string outputPath = Path.Combine(Path.GetTempPath(), "output.jpg");

            // Act & Assert
            Assert.Throws<FileNotFoundException>(() => imageResizer.ResizeImage(nonExistentF
        }

        [Test]
        public void ResizeImage_ShouldCreateResizedImage_WhenInputFileExists()
        {
            // Arrange
            var imageResizer = new ImageResizer();
```

```
            string tempDirectory = Path.GetTempPath();
            string inputPath = Path.Combine(tempDirectory, "input.jpg");
            string outputPath = Path.Combine(tempDirectory, "output.jpg");

            // Create a valid dummy image file
            using (Bitmap bitmap = new Bitmap(200, 200))
            {
                using (Graphics g = Graphics.FromImage(bitmap))
                {
                    g.Clear(Color.White);
                    g.DrawRectangle(Pens.Black, 10, 10, 180, 180);
                }

                bitmap.Save(inputPath, ImageFormat.Jpeg);
            }

            try
            {
                // Act
                imageResizer.ResizeImage(inputPath, outputPath, 100, 100);

                // Assert
                Assert.IsTrue(File.Exists(outputPath));
            }
            finally
            {
                // Cleanup
                File.Delete(inputPath);
                File.Delete(outputPath);
            }
        }
    }
}
```

Sonstige Dateien

Dependencies

- No dependencies found