Лашкин Владислав, гр. 251001

Лабораторная работа №5

**FigureController.cs**

[HttpPost]

[Route("settings")]

public IActionResult Settings(Settings settings)

{

FigureService.EncryptionKey = settings.Key;

return NoContent();

}

[HttpGet]

[Route("save")]

public IActionResult Save()

{

FigureService.SaveChanges();

if (FigureService.EncryptPlugin != null)

{

FigureService.EncryptPlugin.EncryptString(FigureService.FilePath, FigureService.EncryptionKey ?? "1111111111111111");

}

if (FigureService.ZipperPlugin != null)

{

string src = FigureService.FilePath;

string dest = src.Replace(FigureService.Format == SerializationFormat.Xml ? ".xml" : ".json", ".gz");

FigureService.ZipperPlugin.CompressAsync(src, dest, FigureService.Format);

}

return NoContent();

}

[HttpPost]

[Route("open")]

public async Task<IActionResult> Open(IFormFile? file)

{

if (file != null && file.Length > 0)

{

string fileName = Path.GetFileName(file.FileName);

string filePath = Path.Combine(Directory.GetCurrentDirectory(), "UploadedFigures", fileName);

FigureService.FilePath = filePath;

string newPath;

await using (var stream = new FileStream(filePath, FileMode.Create))

{

await file.CopyToAsync(stream);

}

string extension = Path.GetExtension(filePath);

if (extension == ".json" || extension == ".xml")

{

FigureService.InitList(filePath);

return Ok();

}

if (extension == ".gz")

{

if (FigureService.ZipperPlugin != null)

{

FigureService.ZipperPlugin.DecompressAsync(filePath, FigureService.Format);

newPath = FigureService.ZipperPlugin.CreateDecompressPath(filePath, FigureService.Format);

FigureService.InitList(newPath);

return Ok();

}

else

{

return NotFound("Плагин для архивации не установлен");

}

}

if (extension == ".data")

{

if (FigureService.EncryptPlugin != null)

{

await FigureService.EncryptPlugin.DecryptString(filePath, FigureService.EncryptionKey ?? "1111111111111111", FigureService.Format);

newPath = FigureService.EncryptPlugin.CreateDecryptedPath(filePath, FigureService.Format);

FigureService.InitList(newPath);

return Ok();

}

else

{

return NotFound("Плагин для шифрования не установлен");

}

}

return NotFound("Неверный формат плагина или файла");

}

else

{

return BadRequest("File with figures has not been uploaded yet");

}

}

**PluginService.cs**

public static class PluginService

{

public static Assembly LoadPlugin(string path)

{

PluginLoadContext loadContext = new PluginLoadContext(path);

return loadContext.LoadFromAssemblyName(new AssemblyName(Path.GetFileNameWithoutExtension(path)));

}

public static IEncryptor? GetEncryptor(Assembly assembly)

{

Type? encryptorType = assembly.GetTypes().FirstOrDefault(type => typeof(IEncryptor).IsAssignableFrom(type), null);

return encryptorType != null

? Activator.CreateInstance(encryptorType) as IEncryptor

: null;

}

public static IZipper? GetZipper(Assembly assembly)

{

Type? encryptorType = assembly.GetTypes().FirstOrDefault(type => typeof(IZipper).IsAssignableFrom(type), null);

return encryptorType != null

? Activator.CreateInstance(encryptorType) as IZipper

: null;

}

}

**IEncryptor.cs**

namespace PluginBase;

public interface IEncryptor

{

public void EncryptString(string path, string key);

public Task DecryptString(string path, string key, SerializationFormat format);

public string CreateDecryptedPath(string srcPath, SerializationFormat format);

}

**IZipper.cs**

namespace PluginBase;

public interface IZipper

{

public void CompressAsync(string srcPath, string destPath, SerializationFormat format);

public void DecompressAsync(string srcPath, SerializationFormat format);

public string CreateDecompressPath(string srcPath, SerializationFormat format);

}

**Zipper.cs**

using System.IO;

using System.IO.Compression;

using PluginBase;

namespace ZipperPlugin;

public class Zipper : IZipper

{

public void CompressAsync(string srcPath, string destPath, SerializationFormat format)

{

using FileStream srcStream = new FileStream(srcPath, FileMode.Open);

using FileStream destStream = new FileStream(destPath, FileMode.OpenOrCreate);

using GZipStream stream = new GZipStream(destStream, CompressionMode.Compress);

srcStream.CopyTo(stream);

}

public void DecompressAsync(string srcPath, SerializationFormat format)

{

using FileStream srcStream = new FileStream(srcPath, FileMode.Open);

using FileStream destStream = new FileStream(CreateDecompressPath(srcPath, format), FileMode.OpenOrCreate);

using GZipStream stream = new GZipStream(srcStream, CompressionMode.Decompress);

stream.CopyTo(destStream);

}

public string CreateDecompressPath(string srcPath, SerializationFormat format)

{

string extension = format == SerializationFormat.Xml ? "xml" : "json";

string newPath = srcPath.Replace(".gz", "\_decompressed." + extension);

return newPath;

}

}

**Encryptor.cs**

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

using PluginBase;

namespace EncryptorPlugin;

public class Encryptor : IEncryptor

{

public void EncryptString(string path, string key)

{

string src;

using (FileStream fileStream = new FileStream(path, FileMode.Open))

{

StringBuilder builder = new StringBuilder(256);

byte[] buffer = new byte[64];

int i;

while ((i = fileStream.Read(buffer, 0, buffer.Length)) != 0)

{

string data = Encoding.UTF8.GetString(buffer, 0, i);

builder.Append(data);

}

src = builder.ToString();

}

try

{

string dir = Path.GetDirectoryName(path);

string name = Path.GetFileNameWithoutExtension(path);

using (FileStream fileStream = new(Path.Combine(dir, name + "\_encrypted.data"), FileMode.OpenOrCreate))

{

using Aes aes = Aes.Create();

byte[] inKey = Encoding.UTF8.GetBytes(key);

aes.Key = inKey;

byte[] iv = aes.IV;

fileStream.Write(iv, 0, iv.Length);

using CryptoStream cryptoStream = new(fileStream, aes.CreateEncryptor(), CryptoStreamMode.Write);

using StreamWriter encryptWriter = new(cryptoStream);

encryptWriter.WriteLine(src);

}

Console.WriteLine("The file was encrypted.");

}

catch (Exception ex)

{

Console.WriteLine($"The encryption failed. {ex.Message}");

}

}

public async Task DecryptString(string path, string key, SerializationFormat format)

{

try

{

await using FileStream fileStream = new(path, FileMode.Open);

using Aes aes = Aes.Create();

byte[] iv = new byte[aes.IV.Length];

int numBytesToRead = aes.IV.Length;

int numBytesRead = 0;

while (numBytesToRead > 0)

{

int n = fileStream.Read(iv, numBytesRead, numBytesToRead);

if (n == 0) break;

numBytesRead += n;

numBytesToRead -= n;

}

byte[] inKey = Encoding.UTF8.GetBytes(key);

await using CryptoStream cryptoStream = new(fileStream, aes.CreateDecryptor(inKey, iv), CryptoStreamMode.Read);

using StreamReader decryptReader = new(cryptoStream);

string decryptedMessage = await decryptReader.ReadToEndAsync();

path = path.Replace("\_encrypted.data", format == SerializationFormat.Xml ? "\_decrypted.xml" : "\_decrypted.json");

await using FileStream file = new(path, FileMode.OpenOrCreate, FileAccess.Write);

byte[] buffer = Encoding.UTF8.GetBytes(decryptedMessage);

await file.WriteAsync(buffer, 0, buffer.Length);

}

catch (Exception ex)

{

Console.WriteLine($"The decryption failed. {ex}");

}

}

public string CreateDecryptedPath(string srcPath, SerializationFormat format)

{

string extension = format == SerializationFormat.Xml ? "xml" : "json";

string newPath = srcPath.Replace("\_encrypted.data", "\_decrypted." + extension);

return newPath;

}

}