ConnectionString Encryption

Create a Console Application

//using System.Security.Cryptography;

//using ZR.CodeExample.SecureMVC.Helpers;

//namespace ZR.CodeExample.SecureMVC.Helpers

//{

// public static class EncryptionHelper

// {

// private static readonly string EncryptionKey = GenerateRandomKey(256);

// public static string Encrypt(string plainText)

// {

// using (Aes aesAlg = Aes.Create())

// {

// aesAlg.Key = Convert.FromBase64String(EncryptionKey);

// aesAlg.IV = GenerateRandomIV(); // Generate a random IV for each encryption

// aesAlg.Padding = PaddingMode.PKCS7; // Set the padding mode to PKCS7

// ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

// using (MemoryStream msEncrypt = new MemoryStream())

// {

// using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))

// {

// using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))

// {

// swEncrypt.Write(plainText);

// }

// }

// return Convert.ToBase64String(aesAlg.IV.Concat(msEncrypt.ToArray()).ToArray());

// }

// }

// }

// public static string Decrypt(string cipherText)

// {

// byte[] cipherBytes = Convert.FromBase64String(cipherText);

// using (Aes aesAlg = Aes.Create())

// {

// aesAlg.Key = Convert.FromBase64String(EncryptionKey);

// aesAlg.IV = cipherBytes.Take(16).ToArray();

// aesAlg.Padding = PaddingMode.PKCS7; // Set the padding mode to PKCS7

// ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

// using (MemoryStream msDecrypt = new MemoryStream(cipherBytes, 16, cipherBytes.Length - 16))

// {

// using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))

// {

// using (StreamReader srDecrypt = new StreamReader(csDecrypt))

// {

// return srDecrypt.ReadToEnd();

// }

// }

// }

// }

// }

// private static byte[] GenerateRandomIV()

// {

// using (Aes aesAlg = Aes.Create())

// {

// aesAlg.GenerateIV();

// return aesAlg.IV;

// }

// }

// private static string GenerateRandomKey(int keySizeInBits)

// {

// // Convert the key size to bytes

// int keySizeInBytes = keySizeInBits / 8;

// // Create a byte array to hold the random key

// byte[] keyBytes = new byte[keySizeInBytes];

// // Use a cryptographic random number generator to fill the byte array

// using (var rng = new RNGCryptoServiceProvider())

// {

// rng.GetBytes(keyBytes);

// }

// // Convert the byte array to a base64-encoded string for storage

// return Convert.ToBase64String(keyBytes);

// }

// }

//}

//class P

//{

// static void Main()

// {

// string plainText = "I am Ziggy Rafiq from United Kingdom";

// // Encrypt the data using the EncryptionHelper

// string cipherText = EncryptionHelper.Encrypt(plainText);

// // Decrypt the data to retrieve the original content

// string decryptedText = EncryptionHelper.Decrypt(cipherText);

// // Store the encrypted and decrypted data in ViewData for use in your view

// Console.WriteLine(cipherText);

// Console.WriteLine(decryptedText);

// }

//}

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

class ManagedAesSample

{

public static void Main()

{

byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

string Password = "1111111111111111";

string data = @"server=ANAMIKA\SQLSERVER;database=PracticeDatabase2;integrated security=true;TrustServerCertificate=true";

string sre = EncryptAesManaged(data, Password, iv);

Console.WriteLine(sre);

Console.WriteLine(Decrypt(sre, Password, iv));

Console.ReadLine();

}

static string Encrypt(string raw, string password, byte[] iv)

{

byte[] Key = Encoding.UTF8.GetBytes(password);

AesManaged aes = new AesManaged();

aes.Key = Key;

aes.IV = iv;

MemoryStream ms = new MemoryStream();

CryptoStream cs = new CryptoStream(ms, aes.CreateEncryptor(),

CryptoStreamMode.Write);

byte[] inputBytes = Encoding.UTF8.GetBytes(raw);

cs.Write(inputBytes, 0, inputBytes.Length);

cs.FlushFinalBlock();

byte[] encr = ms.ToArray();

return Convert.ToBase64String(encr);

}

static string Decrypt(string raw, string password, byte[] iv)

{

byte[] Key = Encoding.UTF8.GetBytes(password);

AesManaged aes = new AesManaged();

aes.Key = Key;

aes.IV = iv;

MemoryStream ms = new MemoryStream();

CryptoStream cs = new CryptoStream(ms, aes.CreateDecryptor(), CryptoStreamMode.Write);

byte[] inputBytes = Convert.FromBase64String(raw);

cs.Write(inputBytes, 0, inputBytes.Length);

cs.FlushFinalBlock();

byte[] encr = ms.ToArray();

return UTF8Encoding.UTF8.GetString(encr, 0 , encr.Length);

}

}

Run it , get the encrypted value of ConectionString

Put it in appSettings.json file

"ConnectionStrings": {

"MyConnection": "VnPezB6YgEY3j7M/3NcxJLS4i9rZtxmKNCyZy9DYMQln0sTr13XFENBmbZrF5iVG5zvURPdXh4+8gVJ7AwtNV1OmRTa9zvcJaNCs62ozPUmsdtTvePVCOOcIFK7iygl2XjTO7sDo+rMpjK4nMKjbWg=="

},

In Web Api Project,

Add the same file

using System.Security.Cryptography;

using System.Text;

public static class EncryptionHelper

{

static string EncryptAesManaged(string raw, string password, byte[] iv)

{

byte[] Key = Encoding.UTF8.GetBytes(password);

AesManaged aes = new AesManaged();

aes.Key = Key;

aes.IV = iv;

MemoryStream ms = new MemoryStream();

CryptoStream cs = new CryptoStream(ms, aes.CreateEncryptor(),

CryptoStreamMode.Write);

byte[] inputBytes = Encoding.UTF8.GetBytes(raw);

cs.Write(inputBytes, 0, inputBytes.Length);

cs.FlushFinalBlock();

byte[] encr = ms.ToArray();

return Convert.ToBase64String(encr);

}

public static string Decrypt(string raw, string password, byte[] iv)

{

byte[] Key = Encoding.UTF8.GetBytes(password);

AesManaged aes = new AesManaged();

aes.Key = Key;

aes.IV = iv;

MemoryStream ms = new MemoryStream();

CryptoStream cs = new CryptoStream(ms, aes.CreateDecryptor(), CryptoStreamMode.Write);

byte[] inputBytes = Convert.FromBase64String(raw);

cs.Write(inputBytes, 0, inputBytes.Length);

cs.FlushFinalBlock();

byte[] encr = ms.ToArray();

return UTF8Encoding.UTF8.GetString(encr, 0, encr.Length);

}

}

Go to DBContext File

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

string str = \_configuration.GetConnectionString("MyConnection");

byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

string Password = "1111111111111111";

string connection = EncryptionHelper.Decrypt(str, Password, iv);

optionsBuilder.UseSqlServer(connection);

//optionsBuilder.UseSqlServer("server=ANAMIKA\\SQLSERVER;database=PracticeDatabase1;integrated security=true;TrustServerCertificate=true");

}

But the better way to secure your connectionStrings are

Use secret

Use environment variables

Create env variable

setx ASPNETCORE\_ConnectionStrings\_\_h1 server=ANAMIKA\\SQLSERVER;database=PracticeDatabase1;integrated security=true;TrustServerCertificate=true

Add in Program.cs file

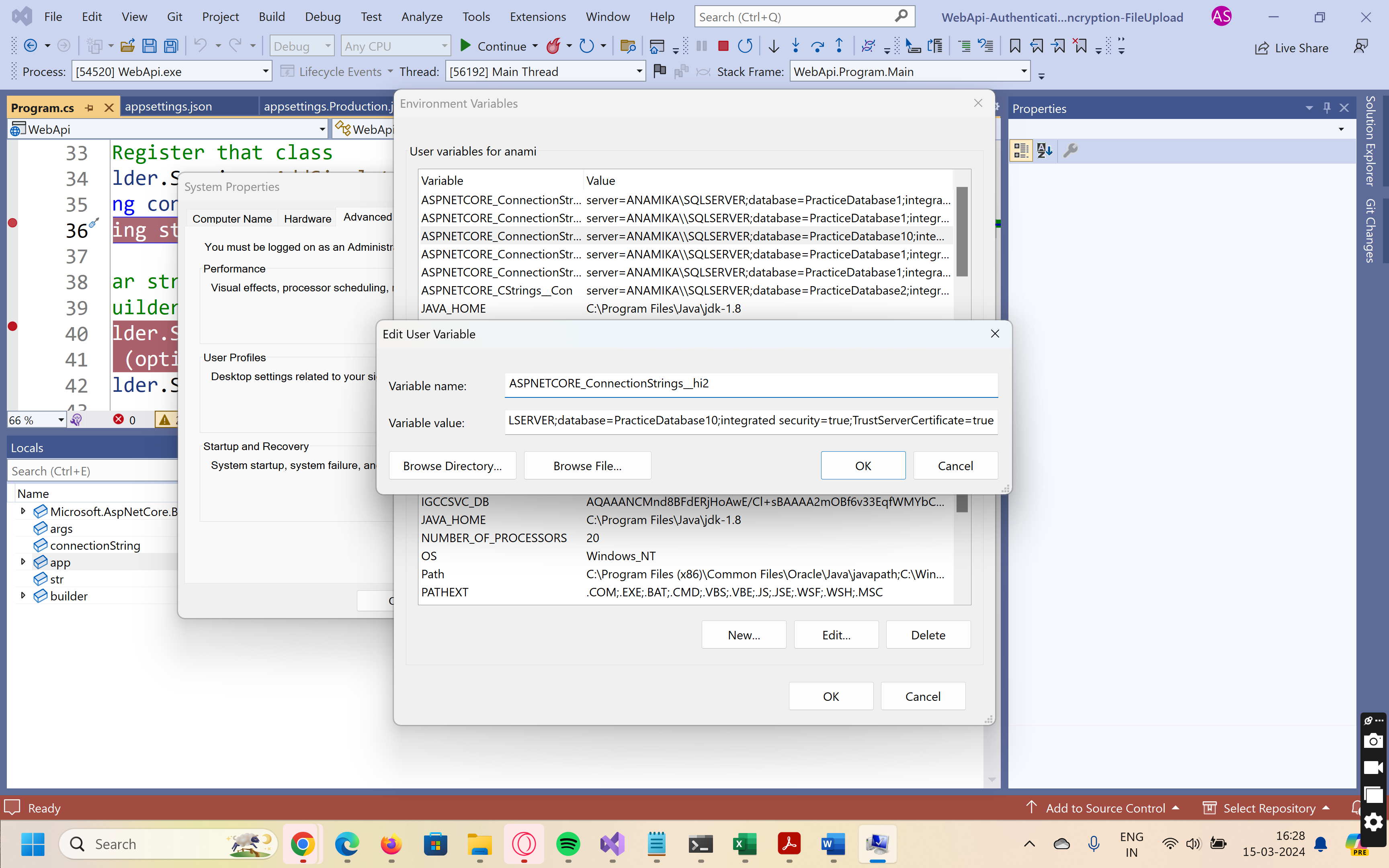
string str = builder.Configuration.GetSection("ConnectionStrings:Hi").Value;

builder.Services.AddDbContext<StudentDbContext>

(options => options.UseSqlServer(str));

builder.Services.AddControllers();

Restart VS



There is no need to store its entry in appSettings.json file

Secrets

{

"cn:cn1":"server=ANAMIKA\\SQLSERVER;database=PracticeDatabase10;integrated security=true;TrustServerCertificate=true"

}

string str1 = builder.Configuration.GetSection("cn:cn1").Value;

builder.Services.AddDbContext<StudentDbContext>

(options => options.UseSqlServer(str1));