Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №3

з дисципліни
«Компоненти програмної інженерії. Якість та тестування програмного забезпечення»

на тему
«Unit тестування з використанням методів White Box Testing»

Виконав:

студент групи ІП-93

Домінський Валентин Олексійович

номер залікової книжки: 9311

номер у списку: 9

Перевірив:

Бабарикін Ігор Владиславович

Київ 2021

Зміст

Мета:

Написати Unit тести з використанням методів White Box Testing

Завдання:

N п/п	9311 mod 6	Library
1	0, 2, 3	IIG.BinaryFlag
2	1, 4, 5	IIG.PasswordHashingUtils

Варіант = 9311 mod 6 = 5, отже провести тестування IIG.PasswordHashingUtils

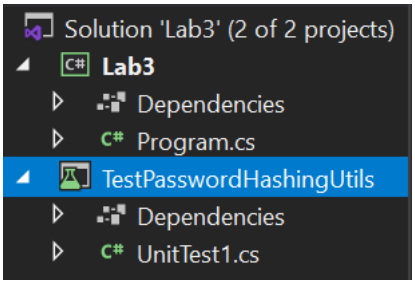
Отже, мій вибір для лабораторної:

- .NET 5
- Бібліотека для тестування xUnit
- Бібліотека IIG.PasswordHashingUtils

Хід роботи:

Початок роботи:

Я створив проект “Lab3” на .NET 5, додав xUnit Test Project “TestPasswordHashingUtils” та бібліотеку IIG.PasswordHashingUtils:



Оскільки у даній лабораторній Ми використовуємо BBT, то можемо переглянути її вміст та зробити відповідні графи для подальшого використовування їх при написанні тестів.

Давайте пройдемося по коду та спробуємо його проаналізувати:

Приватні поля

```
/// <summary>
///     Mod Adler Const for Adler32CheckSum
/// </summary>
private static uint _modAdler32 = 65521;

/// <summary>
///     First Level Salt
/// </summary>
private static string _salt = "put your soul(or salt) here";
```

Як видно з даного шматку, Ми маємо два приватних статичних поля з певними значеннями за замовченням:

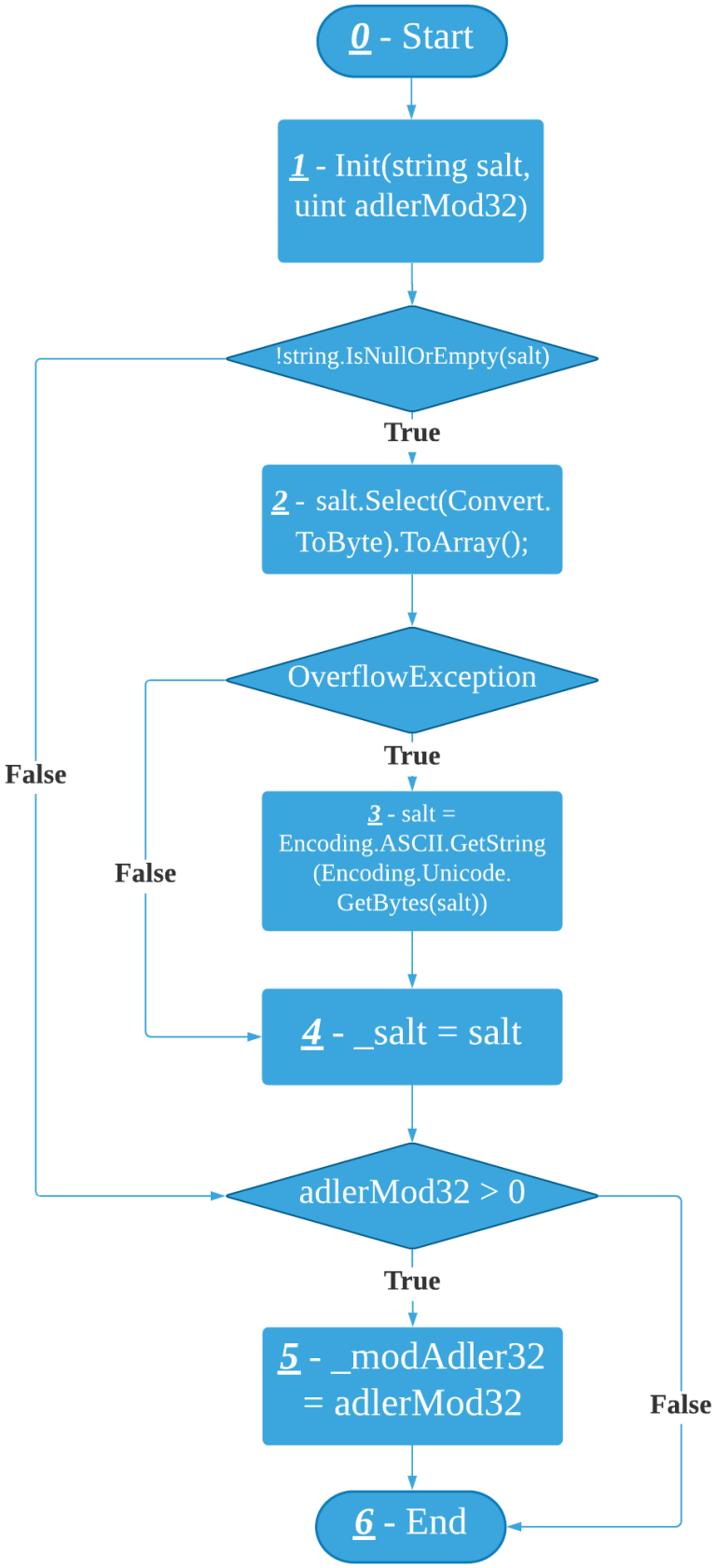
- _modAdler представляє з себе модуль найбільшого простого числа, меншого, ніж 2^{16} , який буде використовуватися при хешуванні паролю
- _salt використовується для створення певного хешу Нашого паролю

Init

```
/// <summary>
///     Init PasswordHasher Parameters
/// </summary>
/// <param name="salt">First Level Salt</param>
/// <param name="adlerMod32">Mod Adler Const for Adler32CheckSum</param>
public static void Init(string salt, uint adlerMod32)
{
    if (!string.IsNullOrEmpty(salt))
    {
        try
        {
            salt.Select(Convert.ToByte).ToArray();
        }
        catch (OverflowException)
        {
            salt = Encoding.ASCII.GetString(Encoding.Unicode.GetBytes(salt));
        }

        _salt = salt;
    }

    if (adlerMod32 > 0)
        _modAdler32 = adlerMod32;
}
```

Дана функція перезаписує значення приватних полів, які були вище, з певними перевірками на нові, які приходять як параметри.

Init

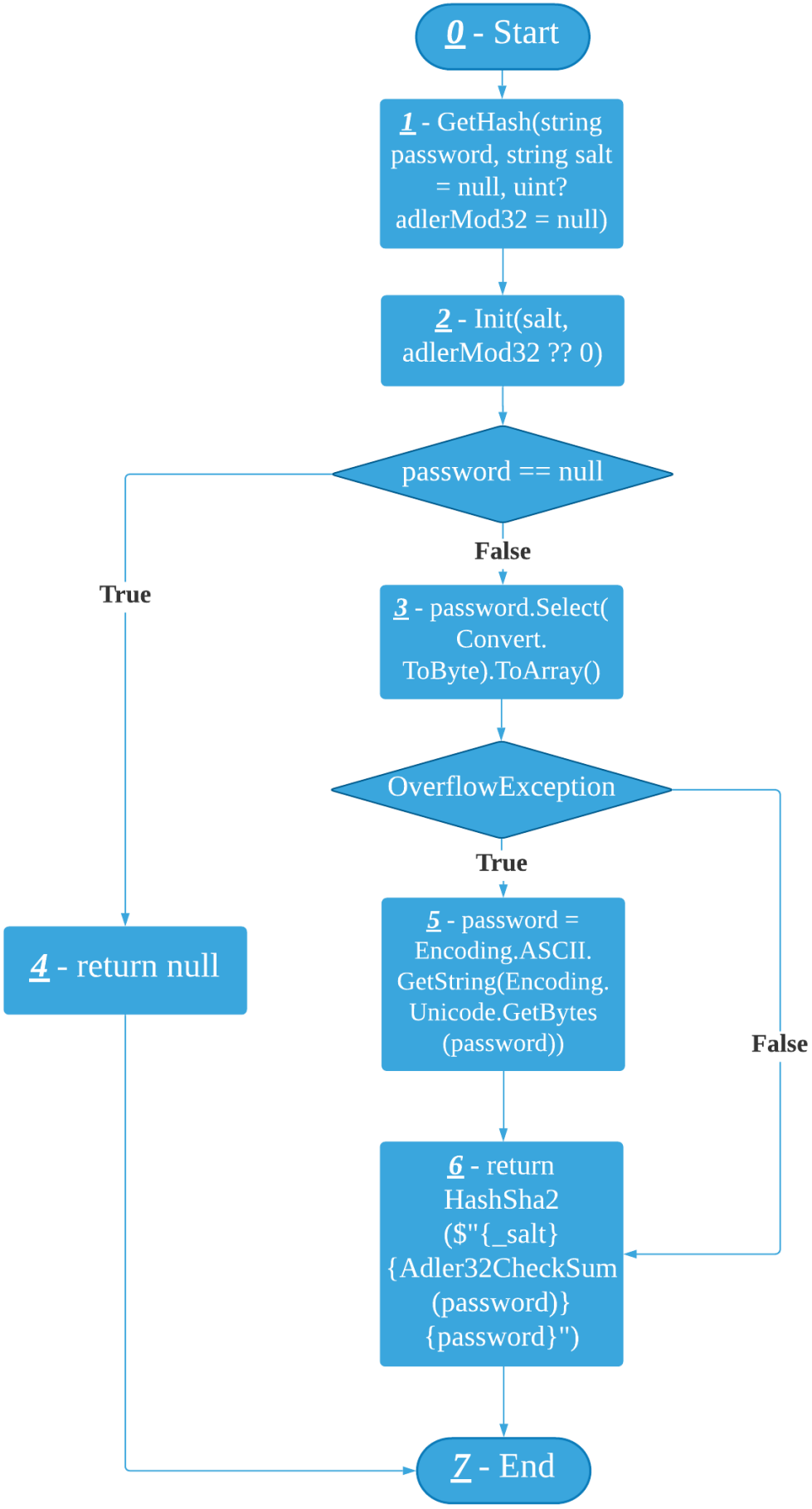

0 - Start
1 - Init(string salt, uint adlerMod32)
!string.IsNullOrEmpty(salt)
True
2 - salt.Select(Convert.ToByte).ToArray();
OverflowException
True
3 - salt = Encoding.ASCII.GetString(Encoding.Unicode.GetBytes(salt))
False
False
4 - _salt = salt
adlerMod32 > 0
True
5 - _modAdler32 = adlerMod32
False
6 - End

GetHash

```
/// <summary>
///     Calculates Hash for Provided String Password
/// </summary>
/// <param name="password">Password</param>
/// <param name="salt">First Level Salt</param>
/// <param name="adlerMod32">Mod Adler Const for Adler32CheckSum</param>
/// <returns>SHA2 String Hash for Provided Password</returns>
public static string GetHash(string password, string salt = null, uint? adlerMod32 = null)
{
    Init(salt, adlerMod32 ?? 0);
    if (password == null)
        return null;
    try
    {
        password.Select(Convert.ToByte).ToArray();
    }
    catch (OverflowException)
    {
        password = Encoding.ASCII.GetString(Encoding.Unicode.GetBytes(password));
    }

    return HashSha2($"{_salt}{Adler32CheckSum(password)}{password}");
}
```

Цей метод повертає певний хеш для паролю, який Ми передали. Сюди ж можна вписати salt та adlerMod, якщо Нам потрібні якісь конкретні значення.

GetHash

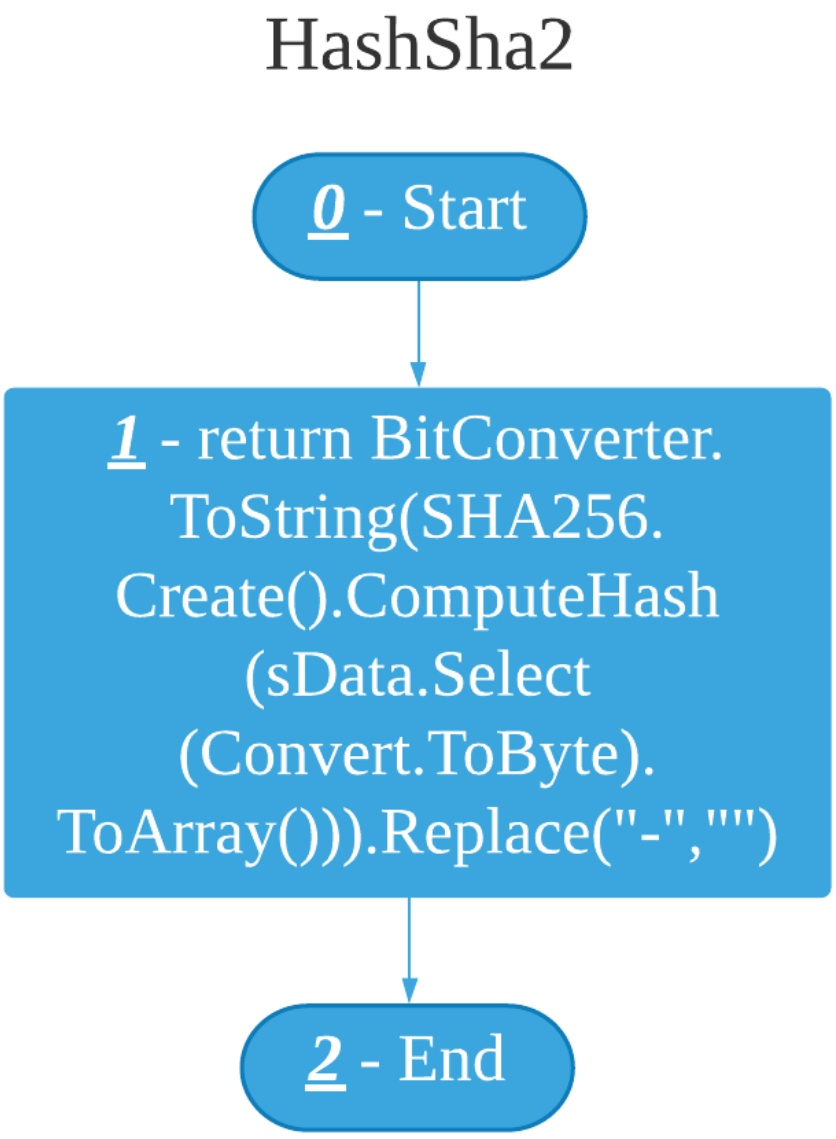

0 - Start
1 - GetHash(string password, string salt = null, uint? adlerMod32 = null)
2 - Init(salt, adlerMod32 ?? 0)
password == null
True
False
3 - password.Select(Convert. ToByte).ToArray()
OverflowException
True
False
4 - return null
5 - password = Encoding.ASCII. GetString(Encoding. Unicode.GetBytes (password))
6 - return HashSha2 ($"{_salt} {Adler32CheckSum (password)} {password}")
7 - End

Далі вже йдуть функції, які є приватними, отже тестувати їх напряму – неможливо

HashSha2

```
/// <summary>
///     Calculates SHA2 Hash for Provided Text
/// </summary>
/// <param name="sData">String Data</param>
/// <returns>String Result of SHA2 Hash for Provided Text</returns>
private static string HashSha2(string sData)
{
    return BitConverter.ToString(SHA256.Create().ComputeHash(sData.Select(Convert.ToByte).ToArray()))
        .Replace("-", "");
}
```

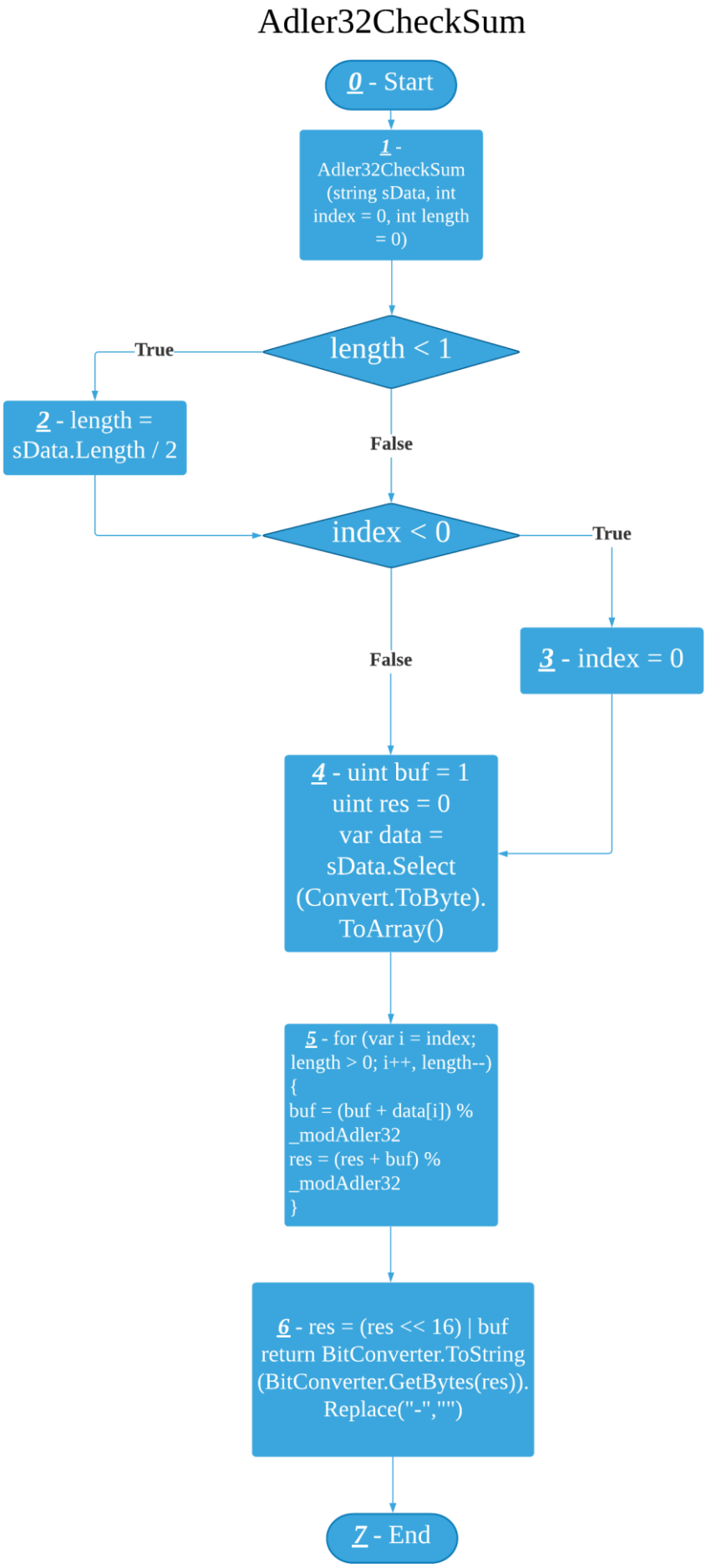
У цьому шматку коду приймається певна стрічка, яка приймає участь у створенні SHA2 хешу.



Adler32CheckSum

```
/// <summary>
///     Calculates Adler32CheckSum for Provided Text and Parameters
/// </summary>
/// <param name="sData">Text</param>
/// <param name="index">Index Adler32CheckSum Parameter</param>
/// <param name="length">Length Adler32CheckSum Parameter</param>
/// <returns>String Representation of Adler32CheckSum Result</returns>
private static string Adler32CheckSum(string sData, int index = 0, int length = 0)
{
    if (length < 1)
        length = sData.Length / 2;
    if (index < 0)
        index = 0;

    uint buf = 1;
    uint res = 0;
    var data = sData.Select(Convert.ToByte).ToArray();

    for (var i = index; length > 0; i++, length--)
    {
        buf = (buf + data[i]) % _modAdler32;
        res = (res + buf) % _modAdler32;
    }

    res = (res << 16) | buf;

    return BitConverter.ToString(BitConverter.GetBytes(res)).Replace("-", "");
}
```

І останній метод Adler32CheckSum. Як видно з XML-коментаря дана функція приймає текст, індекс та довжину Adler32CheckSum, а повертає Adler32CheckSum у вигляді стрічки.



Отже Ми можемо дійти до висновку, що напряму тестувати можна лише дві функції: Init та GetHash.

Тестування:

Init

Execution Route 0_1_6

Почати тестування Я вирішив з методу Init. Але сталася проблема: Я не можу перевірити чи змінилися приватні поля без іншого методу – GetHash. Тобто, якщо повертається один й той же хеш до та після виклику Init, то Ми можемо дійти до висновку, що _modAdler та _salt залишилися тими самими. Давайте спробуємо так і зробити:

```
/// <summary>
/// Test Execution Route 0_1_6 with null and zero
/// Should return true
/// </summary>
[Fact]
| 0 references | VsIG-official, 9 minutes ago | 1 author, 1 change
public void ExecRoute_0_1_6_NullAndZero_True()
{
    // Arrange
    const string SALT = null;
    const uint ADLER_MOD = 0;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.Equal(actual, expected);
}
```

І тест дійсно пройшов! Тепер зробимо те саме, але з іншими значенням, які підуть по цьому ж маршруту:

```
/// <summary>
/// Test Execution Route 0_1_6 with Empty and zero
/// Should return true
/// </summary>
[Fact]
| 0 references | 0 changes | 0 authors, 0 changes
public void ExecRoute_0_1_6_EmptyAndZero_True()
{
    // Arrange
    const string SALT = "";
    const uint ADLER_MOD = 0;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.Equal(actual, expected);
}
```

Також були спроби передати від’ємне значення AdlerMod, але оскільки у бібліотеці використовується uint, який не може приймати негативні значення, то Visual Studio видавало помилку.

Execution Route 0_1_5_6

А тут Я зрозумів одну річ... усі тести будуть використовувати спільні параметри PasswordHasher'у. Тобто, якщо у тесті, який запускається раніше, йде зміна, наприклад, _modAdler32, то вона буде помітна в УСІХ подальших тестах, навіть якщо вони знаходяться у різних класах. Через це Я доволі довгий час не міг зрозуміти, чому падає тест. Вирішилось це за допомогою додаткової функції SetDefaultValues, яка викликається на початку кожного тесту.

```
/// <summary>
/// Sets the default values for PasswordHasher
/// </summary>
4 references | 0 changes | 0 authors, 0 changes
public static void SetDefaultValues()
{
    const string DEFAULT_SALT = "put your soul(or salt) here";
    const uint DEFAULT_ADLER_MOD = 65521;

    PasswordHasher.Init(DEFAULT_SALT, DEFAULT_ADLER_MOD);
}
```

Але давайте перейдемо до самих тестів даного шляху.

```
/// <summary>
/// Test Execution Route 0_1_5_6 with Empty and Positive
/// Should return true
/// </summary>
[Fact]
0 references | VsIG-official, Less than 5 minutes ago | 1 author, 2 changes
public void ExecRoute_0_1_5_6_EmptyAndPositive_True()
{
    // Arrange
    SetDefaultValues();

    const string SALT = "";
    const uint ADLER_MOD = 1;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.NotEqual(actual, expected);
}
```

```
/// <summary>
/// Test Execution Route 0_1_5_6 with null and Positive
/// Should return true
/// </summary>
[Fact]
| 0 references | VsIG-official, Less than 5 minutes ago | 1 author, 2 changes
public void ExecRoute_0_1_5_6_NullAndPositive_True()
{
    // Arrange
    SetDefaultValues();

    const string SALT = null;
    const uint ADLER_MOD = 1;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.NotEqual(actual, expected);
}
```

Якщо глянути на діаграму, то можна зрозуміти, що _adlerMod32 повинен мінятися, оскільки він > 0 (через це й зміниться сам хеш), але не сіль.

Execution Route 0_1_2_4_6

```
/// Test Execution Route 0_1_2_4_6 with not Empty and Zero
/// Should be Not Equal
/// </summary>
[Fact]
| 0 references | VsIG-official, 1 hour ago | 1 author, 1 change
public void ExecRoute_0_1_2_4_6_NotEmptyAndZero_NotEqual()
{
    const string SALT = "Some cool salt";
    const uint ADLER_MOD = 0;
    try
    {
        // Arrange
        SetDefaultValues();

        const string PASSWORD = "Dominskyi";

        string expected = PasswordHasher.GetHash(PASSWORD);

        // Act
        PasswordHasher.Init(SALT, ADLER_MOD);

        string actual = PasswordHasher.GetHash(PASSWORD);

        // Assert
        Assert.NotEqual(actual, expected);
    }
    catch (OverflowException)
    {
        Assert.False(true);
    }
}
```

Даний шлях повинен мати вже сіль, яка дійсно є валідним текстом, на відміну від минулих прикладів, де тестувалися шляхи з передачею порожньої стрічки або взагалі null, але у той же час AdlerMod не повинен мінятися.

Execution Route 0_1_2_3_4_6

```
/// <summary>
/// Test Execution Route 0_1_2_3_4_6 with not Empty and Zero
/// Should be Not Equal
/// </summary>
[Fact]
0 references | 0 changes | 0 authors, 0 changes
public void ExecRoute_0_1_2_3_4_6_NotEmptyAndZero_OverflowExceptionAndNotEqual()
{
    // Arrange
    SetDefaultValues();

    // Let's pretend, that there We have REALLY large string
    const string SALT = "Some REALLY BIG and cool salt";
    const uint ADLER_MOD = 0;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.Throws<OverflowException>(() => PasswordHasher.Init(SALT, ADLER_MOD));
    Assert.NotEqual(actual, expected);
}
```

А тут Я витратив багато часу на написання тесту і все одно не зміг зробити нічого дійсно підходящого. Щоб даний шлях пройти, треба щоб виникнуло OverflowException - виняток, який видається, якщо виконання арифметичної операції, операції приведення до типу або перетворення в контексті, який перевіряється, призводить до переповнення. Оскільки єдина змінна, якою Ми можемо управляти, аби виникнув цей exception – це string salt, то Я намагався записати в неї максимальну можливу к-сть знаків, на кшталт «string salt = new string('A', 2147483647)» або «string salt = new string('A', int.MaxValue)», але при таких значеннях проект не запускався в онлайн редакторах, а на Моїй локальній машині видавало OutOfMemoryException.

Assert.Throws<OverflowException> перевіряє, чи дійсно дана функція видає exception при певних параметрах і, якщо це так, то тест проходить.

Execution Route 0_1_2_3_4_5_6

```
/// <summary>
/// Test Execution Route 0_1_2_3_4_5_6 with not Empty and Not Zero
/// Should be Not Equal
/// </summary>
[Fact]
| 0 references | 0 changes | 0 authors, 0 changes
public void ExecRoute_0_1_2_3_4_5_6_NotEmptyAndNotZero_OverflowExceptionAndNotEqual()
{
    // Arrange
    SetDefaultValues();

    // Let's pretend, that there We have REALLY large string
    const string SALT = "Some REALLY BIG and cool salt";
    const uint ADLER_MOD = 1;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.Throws<OverflowException>(() => PasswordHasher.Init(SALT, ADLER_MOD));
    Assert.NotEqual(actual, expected);
}
```

Цей шлях дуже схожий на попередній, але adlerMod тут уже виставлений > 0, тому й проходимо через усі можливі операції.

Execution Route 0_1_2_4_5_6

```
/// <summary>
/// Test Execution Route 0_1_2_4_5_6 with not Empty and Not Zero
/// Should be Not Equal
/// </summary>
[Fact]
| 0 references | 0 changes | 0 authors, 0 changes
public void ExecRoute_0_1_2_4_5_6_NotEmptyAndNotZero_NotEqual()
{
    // Arrange
    SetDefaultValues();

    const string SALT = "Some cool salt";
    const uint ADLER_MOD = 1;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.NotEqual(actual, expected);
}
```

І останній можливий маршрут, який пов’язаний з методом Init. У даному випадку Ми проходимо через усі стадії без усіляких виключень, при цьому маючи дуже надійний хеш для Нашого паролю.

GetHash

У даній секції Я не буду зачіпати метод Init, оскільки його тестування вже було зроблено

Execution Route 0_1_2_4_7

```
/// <summary>
/// Test Execution Route 0_1_6 with null, null and zero
/// Should be Equal
/// </summary>
[Fact]
✅ | 0 references | 0 changes | 0 authors, 0 changes
public void ExecRoute_0_1_6_Null_NullAndZero_Null()
{
    // Arrange
    SetDefaultValues();

    const string SALT = null;
    const uint ADLER_MOD = 0;
    const string PASSWORD = null;

    // Act
    string actual = PasswordHasher.GetHash(PASSWORD, SALT, ADLER_MOD);

    // Assert
    Assert.Null(actual);
}
```

Перший обраний Мною шлях – найпростіший – перевірка паролю на null, та, якщо він дійсно таким є, return того ж null.

Execution Route 0_1_2_3_6_7

```
/// <summary>
/// Test Execution Route 0_1_2_3_6_7 with not null, not null and positive
/// Should be not Null
/// </summary>
[Fact]
0 references | VslG-official, Less than 5 minutes ago | 1 author, 1 change
public void ExecRoute_0_1_2_3_6_7_NotNullNotNullAndPositive_NotNull()
{
    // Arrange
    SetDefaultValues();

    const string SALT = "Some new cool salt";
    const uint ADLER_MOD = 3;
    const string PASSWORD = "Dominskyi";

    // Act
    string actual = PasswordHasher.GetHash(PASSWORD, SALT, ADLER_MOD);

    // Assert
    Assert.NotNull(actual);
}
```

Це, по суті, найкращий для користувача шлях, адже код повністю відпрацьовує своє, не викликаючи жодних виключень.

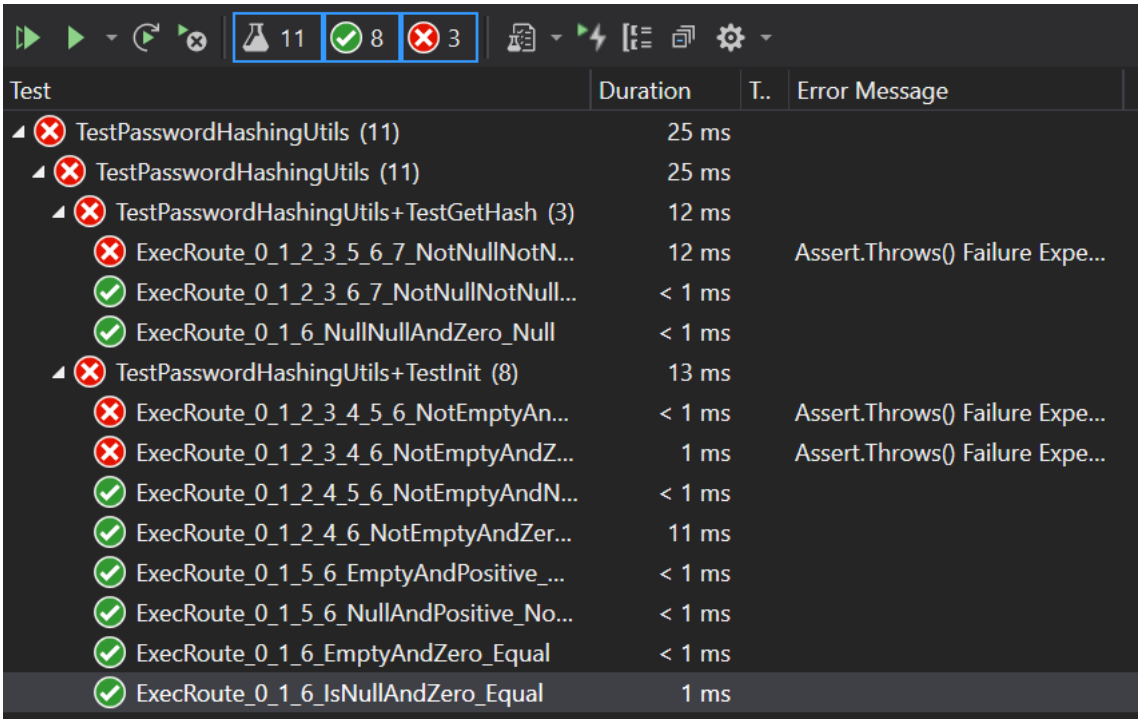
Execution Route 0_1_2_3_5_6_7

```
/// <summary>
/// Test Execution Route 0_1_2_3_5_6_7 with not null, not null and positive
/// Should be not Null
/// </summary>
[Fact]
0 references | 0 changes | 0 authors, 0 changes
public void ExecRoute_0_1_2_3_5_6_7_NotNullNotNullAndPositive_OverflowExceptionAndNotNull()
{
    // Arrange
    SetDefaultValues();

    // Let's pretend, that there We have REALLY large string
    const string SALT = "Some REALLY BIG, new and cool salt";
    const uint ADLER_MOD = 3;
    const string PASSWORD = "Dominskyi";

    // Act
    string actual = PasswordHasher.GetHash(PASSWORD, SALT, ADLER_MOD);

    // Assert
    Assert.Throws<OverflowException>(() => PasswordHasher.Init(SALT, ADLER_MOD));
    Assert.NotNull(actual);
}
```

І останній тест – майже копія минулого, з однією відмінністю – у Нас тут знову OverflowException, який трішки змінює алгоритм.

Результати тестування

Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
8	9,20%	79	90,80%



Як видно з даних результатів, не пройшли успішно лише тести з OverflowException. Саме вони й псують статистику охоплення коду тестами.

Сирцеві коди:

TestPasswordHashingUtils (тести)

```
using System;
using Xunit;
using IIG.PasswordHashingUtils;

namespace TestPasswordHashingUtils
{
  /// <summary>
  /// Class for testing Password Hashing Utils
  /// </summary>
  public class TestPasswordHashingUtils
  {

    /// <summary>
    /// Sets the default values for PasswordHasher
    /// </summary>
    public static void SetDefaultValues()
    {
      const string DEFAULT_SALT = "put your soul(or salt) here";
      const uint DEFAULT_ADLER_MOD = 65521;

      PasswordHasher.Init(DEFAULT_SALT, DEFAULT_ADLER_MOD);
    }

    /*
    Naming:
    1. Execution Route being tested.
    2. The scenario under which it's being tested.
    3. The expected behavior when the scenario is invoked.
    */

    #region ExecutionRoutes

    #region Init

    /// <summary>
    /// Class for testing Init method
    /// </summary>
    public class TestInit
    {

      #region 0_1_6

      /// <summary>
      /// Test Execution Route 0_1_6 with null and zero
      /// Should be  Equal
      /// </summary>
      [Fact]
      public void ExecRoute_0_1_6_IsNullAndZero_Equal()
      {
        // Arrange
        SetDefaultValues();
```

```
    const string SALT = null;
    const uint ADLER_MOD = 0;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.Equal(actual, expected);
}

/// <summary>
/// Test Execution Route 0_1_6 with Empty and zero
/// Should be Equal
/// </summary>
[Fact]
public void ExecRoute_0_1_6_EmptyAndZero_Equal()
{
    // Arrange
    SetDefaultValues();

    const string SALT = "";
    const uint ADLER_MOD = 0;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.Equal(actual, expected);
}

#endregion 0_1_6

#region 0_1_5_6

/// <summary>
/// Test Execution Route 0_1_5_6 with Empty and Positive
/// Should  be not Equal
/// </summary>
[Fact]
public void ExecRoute_0_1_5_6_EmptyAndPositive_NotEqual()
{
    // Arrange
    SetDefaultValues();
```

```
    const string SALT = "";
    const uint ADLER_MOD = 1;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.NotEqual(actual, expected);
}

/// <summary>
/// Test Execution Route 0_1_5_6 with null and Positive
/// Should be not Equal
/// </summary>
[Fact]
public void ExecRoute_0_1_5_6_NullAndPositive_NotEqual()
{
    // Arrange
    SetDefaultValues();

    const string SALT = null;
    const uint ADLER_MOD = 1;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.NotEqual(actual, expected);
}

#endregion 0_1_5_6

#region 0_1_2_4_6

/// <summary>
/// Test Execution Route 0_1_2_4_6 with not Empty and Zero
/// Should be Not Equal
/// </summary>
[Fact]
public void ExecRoute_0_1_2_4_6_NotEmptyAndZero_NotEqual()
{
    SetDefaultValues();

    const string SALT = "Some cool salt";
    const uint ADLER_MOD = 0;
```

```
    try
    {
        // Arrange
        const string PASSWORD = "Dominskyi";

        string expected = PasswordHasher.GetHash(PASSWORD);

        // Act
        PasswordHasher.Init(SALT, ADLER_MOD);

        string actual = PasswordHasher.GetHash(PASSWORD);

        // Assert
        Assert.NotEqual(actual, expected);
    }
    catch (OverflowException)
    {
        Assert.False(true);
    }
}

#endregion 0_1_2_4_6

#region 0_1_2_3_4_6

/// <summary>
/// Test Execution Route 0_1_2_3_4_6 with not Empty and Zero
/// Should be Not Equal
/// </summary>
[Fact]
public void ExecRoute_0_1_2_3_4_6_NotEmptyAndZero_OverflowExceptionAndNotEqual()
{
    // Arrange
    SetDefaultValues();

    // Let's pretend, that there We have REALLY large string
    const string SALT = "Some REALLY BIG and cool salt";
    const uint ADLER_MOD = 0;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.Throws<OverflowException>(() => PasswordHasher.Init(SALT, ADLER_MOD));
    Assert.NotEqual(actual, expected);
}

#endregion 0_1_2_3_4_6
```

```
#region 0_1_2_3_4_5_6

/// <summary>
/// Test Execution Route 0_1_2_3_4_5_6 with not Empty and Not Zero
/// Should be Not Equal
/// </summary>
[Fact]
public void ExecRoute_0_1_2_3_4_5_6_NotEmptyAndNotZero_OverflowExceptionAndNotEqual()
{
    // Arrange
    SetDefaultValues();

    // Let's pretend, that there We have REALLY large string
    const string SALT = "Some REALLY BIG and cool salt";
    const uint ADLER_MOD = 1;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
    Assert.Throws<OverflowException>(() => PasswordHasher.Init(SALT, ADLER_MOD));
    Assert.NotEqual(actual, expected);
}

#endregion 0_1_2_3_4_5_6

#region 0_1_2_4_5_6

/// <summary>
/// Test Execution Route 0_1_2_4_5_6 with not Empty and Not Zero
/// Should be Not Equal
/// </summary>
[Fact]
public void ExecRoute_0_1_2_4_5_6_NotEmptyAndNotZero_NotEqual()
{
    // Arrange
    SetDefaultValues();

    const string SALT = "Some cool salt";
    const uint ADLER_MOD = 1;
    const string PASSWORD = "Dominskyi";

    string expected = PasswordHasher.GetHash(PASSWORD);

    // Act
    PasswordHasher.Init(SALT, ADLER_MOD);

    string actual = PasswordHasher.GetHash(PASSWORD);

    // Assert
```

```
        Assert.NotEqual(actual, expected);
    }

    #endregion 0_1_2_4_5_6

}

#endregion Init

#region GetHash

/// <summary>
/// Class for testing GetHash method
/// </summary>
public class TestGetHash
{

    #region 0_1_6

    /// <summary>
    /// Test Execution Route 0_1_6 with null, null and zero
    /// Should be Null
    /// </summary>
    [Fact]
    public void ExecRoute_0_1_6_NullNullAndZero_Null()
    {
        // Arrange
        SetDefaultValues();

        const string SALT = null;
        const uint ADLER_MOD = 0;
        const string PASSWORD = null;

        // Act
        string actual = PasswordHasher.GetHash(PASSWORD, SALT, ADLER_MOD);

        // Assert
        Assert.Null(actual);
    }

    #endregion 0_1_6

    #region 0_1_2_3_6_7

    /// <summary>
    /// Test Execution Route 0_1_2_3_6_7 with not null, not null and positive
    /// Should be not Null
    /// </summary>
    [Fact]
    public void ExecRoute_0_1_2_3_6_7_NotNullNotNullAndPositive_NotNull()
    {
        // Arrange
        SetDefaultValues();

        const string SALT = "Some new cool salt";
```

```
            const uint ADLER_MOD = 3;
            const string PASSWORD = "Dominskyi";

            // Act
            string actual = PasswordHasher.GetHash(PASSWORD, SALT, ADLER_MOD);

            // Assert
            Assert.NotNull(actual);
        }

        #endregion 0_1_2_3_6_7

        #region 0_1_2_3_5_6_7

        /// <summary>
        /// Test Execution Route 0_1_2_3_5_6_7 with not null, not null and positive
        /// Should be not Null
        /// </summary>
        [Fact]
        public void
ExecRoute_0_1_2_3_5_6_7_NotNullNotNullAndPositive_OverflowExceptionAndNotNull()
        {
            // Arrange
            SetDefaultValues();

            // Let's pretend, that there We have REALLY large string
            const string SALT = "Some REALLY BIG, new and cool salt";
            const uint ADLER_MOD = 3;
            const string PASSWORD = "Dominskyi";

            // Act
            string actual = PasswordHasher.GetHash(PASSWORD, SALT, ADLER_MOD);

            // Assert
            Assert.Throws<OverflowException>(() => PasswordHasher.Init(SALT, ADLER_MOD));
            Assert.NotNull(actual);
        }

        #endregion 0_1_2_3_5_6_7

    }

    #endregion GetHash

    #endregion ExecutionRoutes
  }
}
```

Висновки:

Виконавши цю лабораторну роботу я познайомився з **White Box тестуванням** у цілому, а також використав таку техніку, як **Тестування потоку виконання** - при тестуванні даним видом тест-кейси створюються таким чином, щоб перевірити правильність виконання *максимально можливої кількість шляхів* виконання ПЗ

Джерела:

- Github - https://github.com/VsIG-official/Components-Of-Software-Engineering
- TestPasswordHashingUtils - https://github.com/VsIG-official/Components-Of-Software-Engineering/blob/master/Labs/Lab3/TestPasswordHashingUtils/TestPasswordHashingUtils.cs
- Директорія 3-ої лабораторної роботи - https://github.com/VsIG-official/Components-Of-Software-Engineering/tree/master/Labs/Lab3
- Офіційна документація - https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices#characteristics-of-a-good-unit-test
- Лекція по темі White Box Testing - https://docs.google.com/presentation/d/1UEYTs6OrsO_ssROZNnDG72K5Bc30qL-A/edit#slide=id.p1