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 №4

з дисципліни

«Компоненти програмної інженерії. Якість та тестування програмного забезпечення»

на тему

«Інтеграційне тестування модульної системи»

Виконав:

студент групи ІП-93

Завальнюк Максим Євгенович

Залікова книжка: 9312

Номер у списку: 10

Перевірив:

Бабарикін І. В.

Київ 2021

Мета

Провести інтеграційне тестування модульної системи.

Основні кроки виконання

1. Вибрати бібліотеку для тестування за допомогою остачі від ділення номеру залікової книжки на 6.
2. Створити солюшен до якого додати проєкт бібліотеки та через посилання на .dll додати інші бібліотеки.
3. Розгорнути БД на Microsoft SQL Server'і, використовуючи .bak файл(-и) та додати до проекту клас(-и) для роботи з конкретною(-ими) БД.
4. Якщо в Output є File – то БД використовується лише для Input (project), а Input (.dll) виводиться до та читається з файлу за допомогою IIG.FileWorker.
5. Провести інтеграційне тестування I/O (інтеграційні тести, manual тестування, etc.).
6. Підготувати звіт про виконану роботу, який буде містити опис написаних тестів, сирцеві коди юніт-тестів та/або посилання на GitHub де розміщено проект, результати тестування і їх аналіз.

Виконання роботи

Для початку я обрахував свій варіант – 9312 mod 6 = 0. Отже, мені необхідно тестувати інтеграцію бібліотеки **PasswordHashingUtils** з **FileWorker** та проекту **BinaryFlag** із **БД**.

Буду я працювати із платформою **.NET 5** та бібліотекою для тестування **xUnit**.

Далі я створив проект Lab4. До проекту я під’єднав файл **.dll** необхідної бібліотеки та файли проекту для тестування.

Після цього необхідно встановити MSSQL Express та завантажити *backup* необхідної по варіанту БД.

Отже, у бібліотеці з хешування паролю, яку я тестував раніше, є два способи хешування. Усе залежить від того, чи задаємо ми параметри, чи підставляються вони за замовчуванням. У **FileWorker** є два методи на запис та два методи на зчитування. Тому я буду тестувати різні поєднання цих методів.

У випадку з БД методів було небагато, тому я тестував різні типи об’єкту, які створюються від класу *MultipleBinaryFlag*.

Тепер по вище згаданій теорії я створюю тест-кейси:

```
using IIG.BinaryFlag;
using IIG.CoSFE.DatabaseUtils;
using IIG.FileWorker;
using IIG.PasswordHashingUtils;
using Xunit;

namespace Lab4
{
    /// <summary>
    /// Test integration by writing hashed password into file
    /// </summary>
    public class Test_Integration_With_FileWorker
    {
        /// <summary>
        /// Test simple method Write and simple hash
        /// </summary>
        [Fact]
        public void Write_EmptyHash()
        {
            string password = "password";
            string filePath = "test.txt";
            string newHash = PasswordHasher.GetHash(password);
            bool result = BaseFileWorker.Write(newHash, filePath);
            Assert.True(result);
            string data = BaseFileWorker.ReadAll(filePath);
            Assert.Equal(newHash, data);
        }

        /// <summary>
        /// Test simple method Write and moderate hash
        /// </summary>
        [Fact]
        public void Write_ModerateHash()
        {
            string password = "password";
            string salt = "salt";
            uint adler = 312;
            string filePath = "test.txt";
            string newHash = PasswordHasher.GetHash(password, salt, adler);
            bool result = BaseFileWorker.Write(newHash, filePath);
            Assert.True(result);
            string data = BaseFileWorker.ReadAll(filePath);
            Assert.Equal(newHash, data);
        }

        /// <summary>
        /// Test method TryWrite and simple hash
        /// </summary>
```

```
[Fact]
public void TryWrite_EmptyHash()
{
    string password = "";
    string filePath = "test.txt";
    int attempts = 3;
    string newHash = PasswordHasher.GetHash(password);
    bool result = BaseFileWorker.TryWrite(newHash, filePath, attempts);
    Assert.True(result);
    string data = BaseFileWorker.ReadAll(filePath);
    Assert.Equal(newHash, data);
}

/// <summary>
/// Test method TryWrite and moderate hash
/// </summary>
[Fact]
public void TryWrite_ModerateHash()
{
    string password = "";
    string salt = "salt";
    uint adler = 312;
    string filePath = "test.txt";
    int attempts = 3;
    string newHash = PasswordHasher.GetHash(password, salt, adler);
    bool result = BaseFileWorker.TryWrite(newHash, filePath, attempts);
    Assert.True(result);
    string data = BaseFileWorker.ReadAll(filePath);
    Assert.Equal(newHash, data);
}

/// <summary>
/// Test method TryWrite with 0 attempts to write data
/// </summary>
[Fact]
public void ZeroTriesWrite_ModerateHash()
{
    string password = "";
    string salt = "salt";
    uint adler = 312;
    string filePath = "test.txt";
    int attempts = 0;
    string newHash = PasswordHasher.GetHash(password, salt, adler);
    bool result = BaseFileWorker.TryWrite(newHash, filePath, attempts);
    Assert.False(result);
}

/// <summary>
/// Test method ReadLines and simple hash
/// </summary>
```

```
        [Fact]
        public void ReadLines_EmptyHash()
        {
            string password = "\n\r\b\\\\/";
            string filePath = "test.txt";
            int attempts = 3;
            string newHash = PasswordHasher.GetHash(password);
            bool result = BaseFileWorker.TryWrite(newHash, filePath, attempts);
            Assert.True(result);
            string data = BaseFileWorker.ReadLines(filePath)[0];
            Assert.Equal(newHash, data);
        }

        /// <summary>
        /// Test method ReadLines and moderate hash
        /// </summary>
        [Fact]
        public void ReadLines_ModerateHash()
        {
            string password = "\n\r\b\\\\/";
            string salt = "salt";
            uint adler = 312;
            string filePath = "test.txt";
            int attempts = 3;
            string newHash = PasswordHasher.GetHash(password, salt, adler);
            bool result = BaseFileWorker.TryWrite(newHash, filePath, attempts);
            Assert.True(result);
            string data = BaseFileWorker.ReadLines(filePath)[0];
            Assert.Equal(newHash, data);
        }
    }

    /// <summary>
    /// Test integration by storing flags in database
    /// </summary>
    public class Test_Integration_With_DB
    {
        /// <summary>
        /// Config for database
        /// </summary>
        private const string Server = @"GOVERLA2\SQLEXPRESS";
        private const string BinaryFlagDatabase = @"IIG.CoSWE.FlagpoleDB";
        private const bool IsTrusted = true;
        private const string Login = @"coswe";
        private const string Password = @"L}EjpfCgru9X@GLj";
        private const int ConnectionTimeout = 75;
        static readonly FlagpoleDatabaseUtils binaryFlagDatabase = new(Server, Bi
naryFlagDatabase, IsTrusted, Login, Password, ConnectionTimeout);

        /// <summary>
```

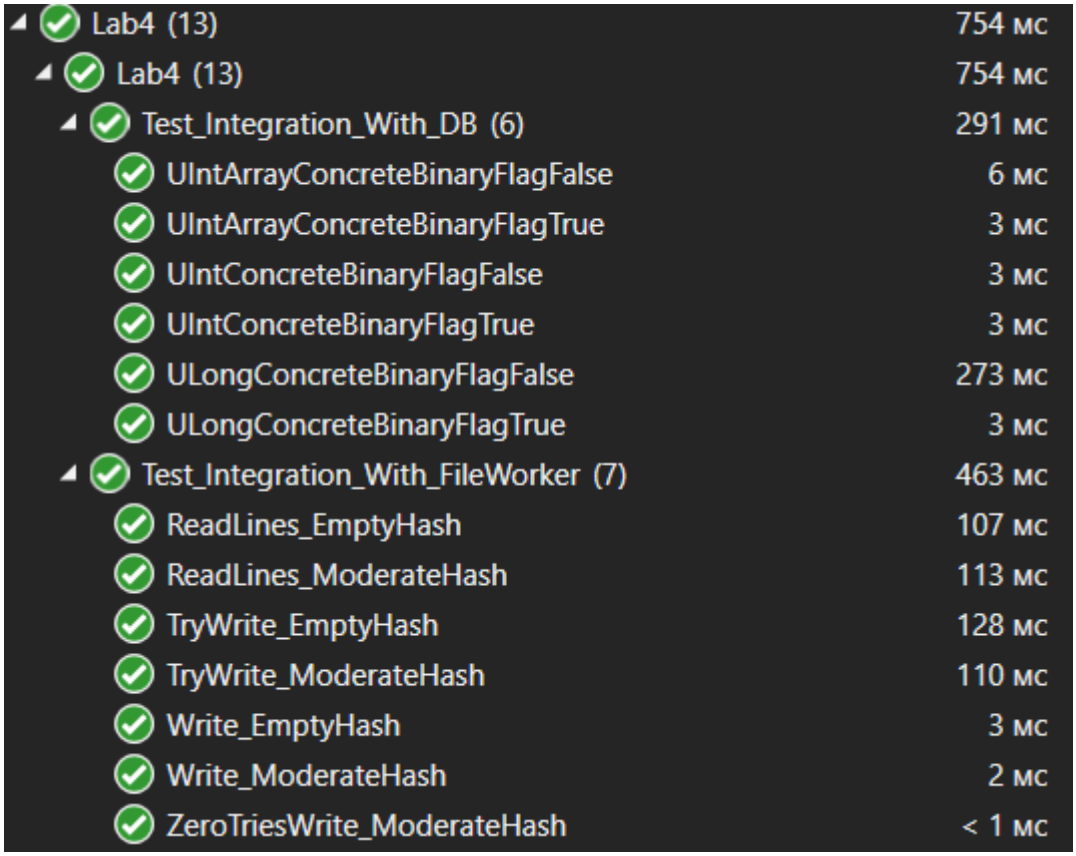
```
        /// Test UIntConcreteBinaryFlag with True value
        /// </summary>
        [Fact]
        public void UIntConcreteBinaryFlagTrue()
        {
            MultipleBinaryFlag actualFlag = new(2, true);
            binaryFlagDatabase.AddFlag(actualFlag.ToString(), (bool)actualFlag.Ge
tFlag());
            int? flagIDNew = binaryFlagDatabase.GetIntBySql("SELECT MAX(MultipleB
inaryFlagID) FROM MultipleBinaryFlags");
            Assert.NotNull(flagIDNew);
            bool newFlag = binaryFlagDatabase.GetFlag((int)flagIDNew, out string
flagView, out bool? flagValue);
            Assert.True(newFlag);
            Assert.Equal(flagView, actualFlag.ToString());
            Assert.Equal(flagValue, actualFlag.GetFlag());
            binaryFlagDatabase.ExecSql("DELETE FROM MultipleBinaryFlags WHERE Mul
tipleBinaryFlagID=" + flagIDNew);
            int? flagIDEmpty = binaryFlagDatabase.GetIntBySql("SELECT MAX(Multipl
eBinaryFlagID) FROM MultipleBinaryFlags");
            Assert.Null(flagIDEmpty);
        }

        /// <summary>
        /// Test UIntConcreteBinaryFlag with False value
        /// </summary>
        [Fact]
        public void UIntConcreteBinaryFlagFalse()
        {
            MultipleBinaryFlag actualFlag = new(2, false);
            binaryFlagDatabase.AddFlag(actualFlag.ToString(), (bool)actualFlag.Ge
tFlag());
            int? flagIDNew = binaryFlagDatabase.GetIntBySql("SELECT MAX(MultipleB
inaryFlagID) FROM MultipleBinaryFlags");
            Assert.NotNull(flagIDNew);
            bool newFlag = binaryFlagDatabase.GetFlag((int)flagIDNew, out string
flagView, out bool? flagValue);
            Assert.True(newFlag);
            Assert.Equal(flagView, actualFlag.ToString());
            Assert.Equal(flagValue, actualFlag.GetFlag());
            binaryFlagDatabase.ExecSql("DELETE FROM MultipleBinaryFlags WHERE Mul
tipleBinaryFlagID=" + flagIDNew);
            int? flagIDEmpty = binaryFlagDatabase.GetIntBySql("SELECT MAX(Multipl
eBinaryFlagID) FROM MultipleBinaryFlags");
            Assert.Null(flagIDEmpty);
        }

        /// <summary>
        /// Test ULongConcreteBinaryFlag with True value
        /// </summary>
```

```
        [Fact]
        public void ULongConcreteBinaryFlagTrue()
        {
            MultipleBinaryFlag actualFlag = new(65 - 1, true);
            binaryFlagDatabase.AddFlag(actualFlag.ToString(), (bool)actualFlag.Ge
tFlag());
            int? flagIDNew = binaryFlagDatabase.GetIntBySql("SELECT MAX(MultipleB
inaryFlagID) FROM MultipleBinaryFlags");
            Assert.NotNull(flagIDNew);
            bool newFlag = binaryFlagDatabase.GetFlag((int)flagIDNew, out string
flagView, out bool? flagValue);
            Assert.True(newFlag);
            Assert.Equal(flagView, actualFlag.ToString());
            Assert.Equal(flagValue, actualFlag.GetFlag());
            binaryFlagDatabase.ExecSql("DELETE FROM MultipleBinaryFlags WHERE Mul
tipleBinaryFlagID=" + flagIDNew);
            int? flagIDEmpty = binaryFlagDatabase.GetIntBySql("SELECT MAX(Multipl
eBinaryFlagID) FROM MultipleBinaryFlags");
            Assert.Null(flagIDEmpty);
        }

        /// <summary>
        /// Test ULongConcreteBinaryFlag with False value
        /// </summary>
        [Fact]
        public void ULongConcreteBinaryFlagFalse()
        {
            MultipleBinaryFlag actualFlag = new(65 - 1, false);
            binaryFlagDatabase.AddFlag(actualFlag.ToString(), (bool)actualFlag.Ge
tFlag());
            int? flagIDNew = binaryFlagDatabase.GetIntBySql("SELECT MAX(MultipleB
inaryFlagID) FROM MultipleBinaryFlags");
            Assert.NotNull(flagIDNew);
            bool newFlag = binaryFlagDatabase.GetFlag((int)flagIDNew, out string
flagView, out bool? flagValue);
            Assert.True(newFlag);
            Assert.Equal(flagView, actualFlag.ToString());
            Assert.Equal(flagValue, actualFlag.GetFlag());
            binaryFlagDatabase.ExecSql("DELETE FROM MultipleBinaryFlags WHERE Mul
tipleBinaryFlagID=" + flagIDNew);
            int? flagIDEmpty = binaryFlagDatabase.GetIntBySql("SELECT MAX(Multipl
eBinaryFlagID) FROM MultipleBinaryFlags");
            Assert.Null(flagIDEmpty);
        }

        /// <summary>
        /// Test UIntArrayConcreteBinaryFlagFalse with True value
        /// </summary>
        [Fact]
        public void UIntArrayConcreteBinaryFlagTrue()
```

```
        {
            MultipleBinaryFlag actualFlag = new(65 + 1, true);
            binaryFlagDatabase.AddFlag(actualFlag.ToString(), (bool)actualFlag.Ge
tFlag());
            int? flagIDNew = binaryFlagDatabase.GetIntBySql("SELECT MAX(MultipleB
inaryFlagID) FROM MultipleBinaryFlags");
            Assert.NotNull(flagIDNew);
            bool newFlag = binaryFlagDatabase.GetFlag((int)flagIDNew, out string
flagView, out bool? flagValue);
            Assert.True(newFlag);
            Assert.Equal(flagView, actualFlag.ToString());
            Assert.Equal(flagValue, actualFlag.GetFlag());
            binaryFlagDatabase.ExecSql("DELETE FROM MultipleBinaryFlags WHERE Mul
tipleBinaryFlagID=" + flagIDNew);
            int? flagIDEmpty = binaryFlagDatabase.GetIntBySql("SELECT MAX(Multipl
eBinaryFlagID) FROM MultipleBinaryFlags");
            Assert.Null(flagIDEmpty);
        }

        /// <summary>
        /// Test UIntArrayConcreteBinaryFlagFalse with False value
        /// </summary>
        [Fact]
        public void UIntArrayConcreteBinaryFlagFalse()
        {
            MultipleBinaryFlag actualFlag = new(65 + 1, false);
            binaryFlagDatabase.AddFlag(actualFlag.ToString(), (bool)actualFlag.Ge
tFlag());
            int? flagIDNew = binaryFlagDatabase.GetIntBySql("SELECT MAX(MultipleB
inaryFlagID) FROM MultipleBinaryFlags");
            Assert.NotNull(flagIDNew);
            bool newFlag = binaryFlagDatabase.GetFlag((int)flagIDNew, out string
flagView, out bool? flagValue);
            Assert.True(newFlag);
            Assert.Equal(flagView, actualFlag.ToString());
            Assert.Equal(flagValue, actualFlag.GetFlag());
            binaryFlagDatabase.ExecSql("DELETE FROM MultipleBinaryFlags WHERE Mul
tipleBinaryFlagID=" + flagIDNew);
            int? flagIDEmpty = binaryFlagDatabase.GetIntBySql("SELECT MAX(Multipl
eBinaryFlagID) FROM MultipleBinaryFlags");
            Assert.Null(flagIDEmpty);
        }
    }
}
```

Тестів вийшло небагато, але вони показують основу інтеграції різних систем та перевіряють її працездатність.

Результат виконання:



Висновок

Під час виконання роботи я познайомився із технологією **інтеграційних** тестів. Можна сказати, що така робота підсумовує попередні лабораторні роботи.

Джерела

1. Папка з програмою.