Eseményvezérelt alkalmazások: 6. gyakorlat

Szavak számlálásának tesztelése

Készítsünk egységtesztelést a korábban megírt `WordCount` projekt `Statistics` osztályához.

Perzisztencia leválasztása

Szervezzük ki a fájl beolvasását egy külön `Persistence` projektbe.

Adjunk hozzá egy új projektet a korábbi `Solution`-hoz a jobb klikk `New project` gomb segítségével. Az új projekt létrehozása ablakban válasszuk ki a **Class Library** projektsablont.

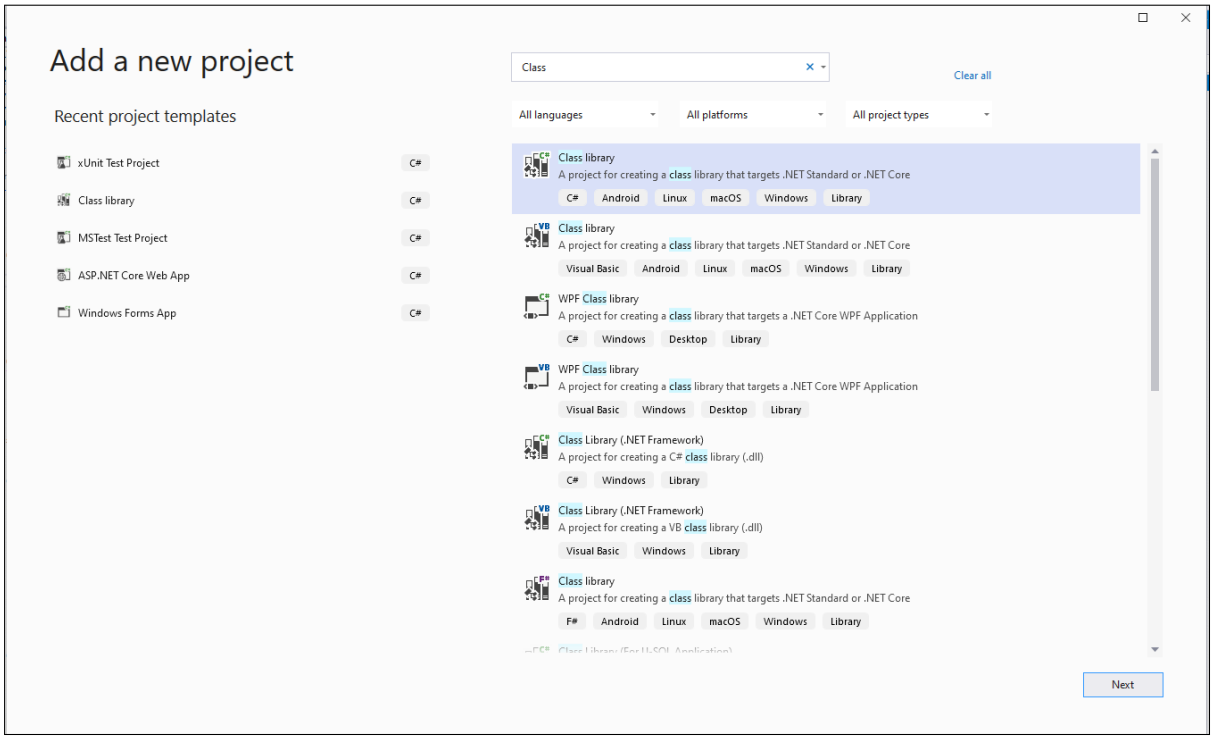

Figure 1: Új Class library projekt létrehozása

Hozzunk létre egy `IFileManager` interfészt, amely tartalmazzon egy `string Load(string path)` metódust. Az interfészt valósítsa meg egy `FileManager` osztály. A `Load` függvény olvassa be a paraméterben kapott fájl tartalmát és adja vissza azt.

Az így létrehozott funkcionalitást függőségi befecskendezés (*Dependency Injection*) segítségével hivatkozzuk a `Statistics` osztályban.

```
private readonly IFileManager _fileManager;

public Statistics(IFileManager fileManager)
{
```

```
    _fileManager = fileManager;
}
```

A `Load` függvényt alakítsuk át úgy, hogy a `FileManager` osztály `Load` függvényét hívja.

A *Dependency Injection* segítségével egy osztályba olyan más osztályt vagy osztályokat "fecskendezhetünk be", amelytől az adott osztály működése függ. Ilyenkor ezeknek az osztályoknak a példányosítása a függőséget felhasználó osztálytól függetlenül történik. Ezt nevezzük **Inversion of Control** paradigmának is.

Ahhoz, hogy a *Dependency Injection* működjön, a `Statictics` osztály meghívásának helyén példányosítunk egy `IFileManager` objektumot:

```
private readonly IFileManager _fileManager;

IFileManager fileManager = new FileManager();
var statistics = new Statistics(fileManager)
```

A `Persistence` osztály használatához ne felejtsük el a referenciát beállítani a `WordCounter` projektünkben. A `Dependency Injection` használatához telepítsük az `Microsoft.Extensions.DependencyInjection` package-t.

Kitekintés az IoC tárolókra *(opcionális)*
Összetettebb, sok függőség befecskendezését igénylő alkalmazásoknál a .NET keretrendszerre is bízhatjuk a példányosítást.
Az *IoC tároló* (*IoC container*) egy olyan *Inversion of Control* paradigmájú komponens, amely lehetőséget ad szolgáltatások megvalósításának dinamikus (futási idejű) betöltésére:

- egy központi regisztráció, amelyet minden programkomponens elérhet, és felhasználhat;
- a típusokat (elsősorban) interfész alapján azonosítja, és az interfészhez csatolja a megvalósító osztályt a tárolóba történő regisztrációkor megadjuk a szolgáltatás interfészét és megvalósításának típusát (vagy példányát);
- a szolgáltatást interfész alapján kérjük le, ekkor példányosul a szolgáltatás vagy kapunk egy már létező példányt;
- amennyiben a szolgáltatásnak függősége van, a tároló azt is példányosítja.

A .NET keretrendszerben az IoC tároló paradigmájára a `ServiceCollection` osztály ad implementációt, amelynek egy példányába mindkét osztályt (`Statistics`, `FileManager`) regisztrálnunk kell. Ilyenkor nem szükséges "manuálisan" példányosítanunk sem a `FileManager`, sem pedig a `Statistics` osztályunkat, helyette a regisztrált osztályokat a `serviceProvider.GetService<T>()` függvény segítségével érhetjük el. Ilyenkor az adott osztály függőségei is automatikusan injektálásra kerülnek.

```
var serviceProvider = new ServiceCollection()
                .AddTransient<Statistics>()
                .AddTransient<IFileManager, FileManager>()
                .BuildServiceProvider();

Statistics statistics = serviceProvider.GetService<Statistics>();
```

Statistics osztály kiegészítése

Egészítsük ki a `Statistics` osztályt az alábbi funkcionalitással:

1. `public void ClearWord(string word)`: A paraméterben kapott string elejéről, illetve végéről kitörli a nem betű karaktereket. A `CountWord()` függvényt refaktoráljuk úgy, hogy ezt a metódust hívja a szavak tisztítására.
2. `public void FilterByLength(int minLength)`: Módosítsa úgy a `WordCount` propertyt, hogy csak azok a kulcsok maradjanak, amelyek hosszúsága nagyobb vagy egyenlő a paraméterben kapott számnál.
3. `public void FilterByOccurence(int minOccurence)`: Módosítsa úgy a `WordCount` propertyt, hogy csak azok a kulcsok maradjanak, amelyhez tartozó előfordulás nagyobb vagy egyenlő a

paraméterben kapott számnál.

Tesztelés megvalósítása

MSTest

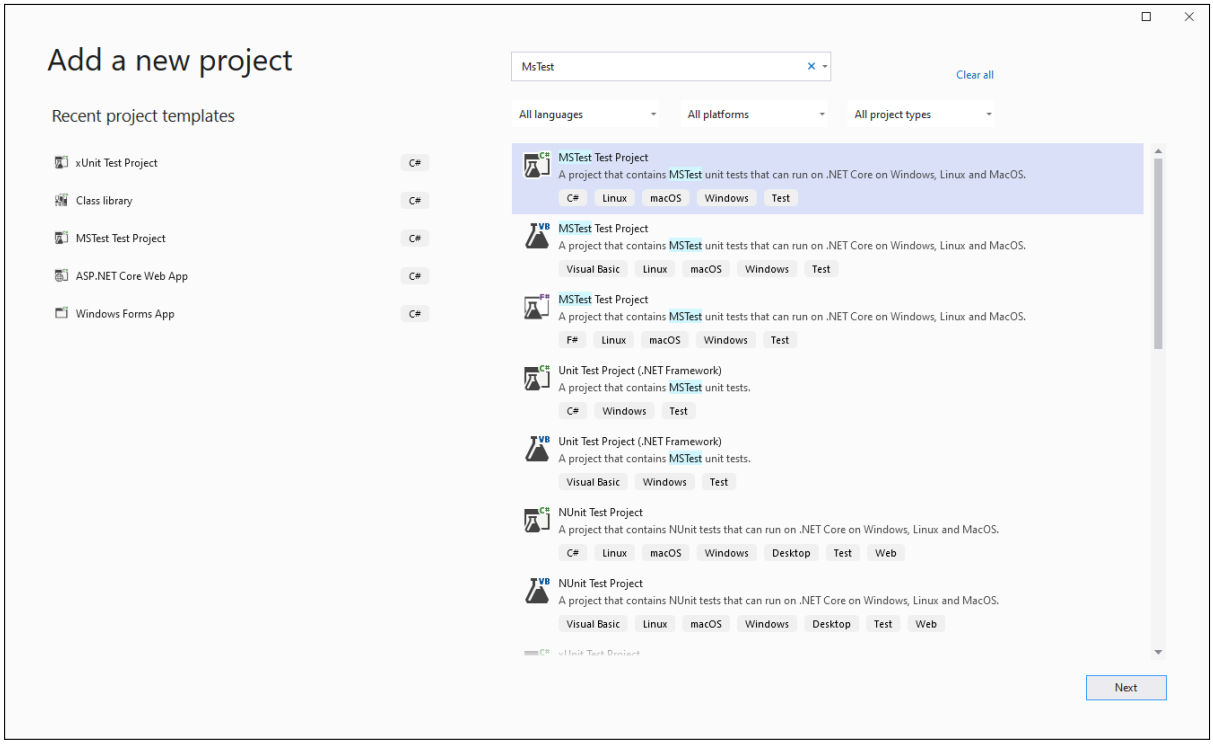

Figure 2: Új MSTest projekt létrehozása

Adjunk hozzá egy új teszt projektet a korábbi `Solution`-hoz a jobb klikk `New project` gomb segítségével. Az új projekt létrehozása ablakban válasszuk ki a **MSTest** projektsablont.

A létrejött osztályban valósítsuk meg a `Statistics` osztály tesztelését.

Az osztály fölött láthatjuk a `[TestClass]` annotációt. Ennek segítségével tudjuk jelezni, hogy az adott osztály teszteseteket fog tartalmazni. Az egyes teszt metódusokat ahhoz, hogy futtatni tudjuk, a `[TestMethod]` annotációval kell ellátnunk.

Az egyes teszteket a *jobb klikk -> Run test/Debug test*, vagy pedig a *View -> Text Explorer* megnyitásával tudjuk futtatni.

Teszteljük az alábbi eseteket:

`ClearWord` függvény tesztjei:

1. A szó elején találhatók nem betű karakterek megfelelően eltávolításra kerülnek?
2. A szó végén találhatók nem betű karakterek megfelelően eltávolításra kerülnek?
3. A szó elején és végén együttesen is megfelelően eltávolításra kerülnek-e?
4. A szó közepén található nem kerül eltávolításra?

Ahhoz, hogy ellenőrizzük, hogy a működés helyes volt-e az `Assert` osztály statikus függvényeit használjuk. A különböző függvények segítségével lehetőségünk van ellenőrizni, hogy a kapott eredmény megegyezik-e a várttal, egy megadott feltétel teljesül-e vagy sem, stb. Amennyiben az adott vizsgálat nem teljesül vagy hibára fut, úgy egy `AssertFailedException` kivétel fog keletkezni, amely hatására a teszt futása sikertelen lesz.

Az egyes tesztek futásának eredményét láthatjuk a *Text Explorerben*, illetve a függvény mellett megjelenő pipa vagy `x` alapján is.

A `Statistics` osztály további függvényeinek teszteléséhez *mockoljuk* ki a fájlbetöltést. Ezt a `Moq` package segítségével tudjuk megtenni, amely lehetőséget ad arra, hogy egy interfész megvalósítását egyszerű, hibamentes funkcionalitással imitáljuk.

A `Statistics` osztály példányosításakor az `IFileManager` típusú paraméter legyen egy, a `Moq` package segítségével példányosított osztály.

```
private readonly Statistics _statisctics;
private readonly Mock<IFileManager> _mock;

public TestStatistics()
{
   _mock = new Mock<IFileManager>();
   _statisctics = new Statistics(_mock.Object);

}
```

A `Moq` segítségével lehetőségünk van arra, hogy az egyes függvények működését és visszatérési értékét tetszőlegesen felüldefiniáljuk.

Például:

```
_mock.Setup(m => m.Load("path")).Returns("test");
```

Ebben az esetben minden olyan helyen, ahol a `Load` függvény a *"path"* bemeneti paraméterrel kerül meghívásra, úgy a függvény visszatérési értéke a *"test"* string lesz.

Ennek segítségével teszteljük le az alábbi eseteket:

Az `IFileManager` interfész `Load` függvényének mockolásával teszteljük a `Statistics` osztály `Load` függvényét:

1. A függvény meghívása után a `FileContent` property a mockolt stringet tartalmazza-e?
2. A `Moq` segítségével szimulálhatjuk azt is, hogy a függvény kivételt dob. Írjunk erre is egy tesztesetet.

Miután meggyőződtünk róla, hogy a `Load` függvényünk jól működik, ellenőrizzük a `CountWords()` függvényt is.

Ellenőrizzük az alábbiakat:

1. Üres szó esetén a dictionary is üres-e?
2. Amennyiben csak nem betű karaktereket tartalmazó szavak szerepelnek, üres-e a dictionary?
3. Amennyiben ugyanaz a szó ismétlődik többször, bekerül-e a dictionary-be a helyes elemszámmal?
4. Amennyiben ugyanaz a szó ismétlődik nem betű karaktereket tartalmazva, bekerül-e a dictionary-be a helyes elemszámmal?
5. Amennyiben ugyanaz a szó ismétlődik kis és nagybetűvel is, bekerül-e a dictionary-be a helyes elemszámmal?
6. Amennyiben ugyanaz több különböző ismétlődik, bekerül-e az összes a helyes elemszámmal?

> Megjegyzés: A teszteseteket érdemes úgy felépíteni, hogy az egyszerű esetből haladunk a bonyolultabb felé, könnyítve ezáltal a hibák kiszűrését (ha már az egyszerű sem működik, a komplexebb sem fog). Érdemes továbbá úgy elnevezni az egyes teszt metódusokat, hogy következtetni lehessen a tesztelni kívánt működésre.

Annak érdekében, hogy biztosítsuk azt, hogy a `WordCount`, illetve `FileContent` property-ben ne maradhasson bent valamely korábban futtattot teszt eredménye, valósítsunk meg a `[TestCleanUp]` annotációval ellátott metódust. Az ebben a metódusban definiált funkcionalitás minden teszt futása után végre fog hajtódni (akkor is, ha csoportosan futtatunk teszteket). Amennyiben azt szeretnénk, hogy egy funkcionalitás minden teszt futása előtt történjen meg, úgy a `[TestInitialize]` annotációt kell használnunk.

Ellenőrizzük a szavak előfordulása szerinti szűrést is az alábbi esetekre:

1. Ha egyik kulcshoz tartozó érték sem felel meg az előfordulás számnak, akkor üres lesz-e a dictionary?
2. Ha pontos egyezés van az előfordulással, bent marad-e a dictionary-ben az adott kulcs?

3. Ha az előfordulás nagyobb, bent marad-e a dictionary-ben?

Ellenőrizzük a szavak hossza szerinti szűrést is az alábbi esetekre:

1. Ha egyik kulcs hosszúsága sem felel meg a hosszúságnak, akkor üres lesz-e a dictionary?
2. Ha pontos egyezés van a hosszúsággal, bent marad-e a dictionary-ben az adott kulcs?
3. Ha adott kulcs hosszúsága nagyobb, bent marad-e a dictionary-ben?

XUnit

A fenti teszteket valósítsuk meg az `XUnit` segítségével is.

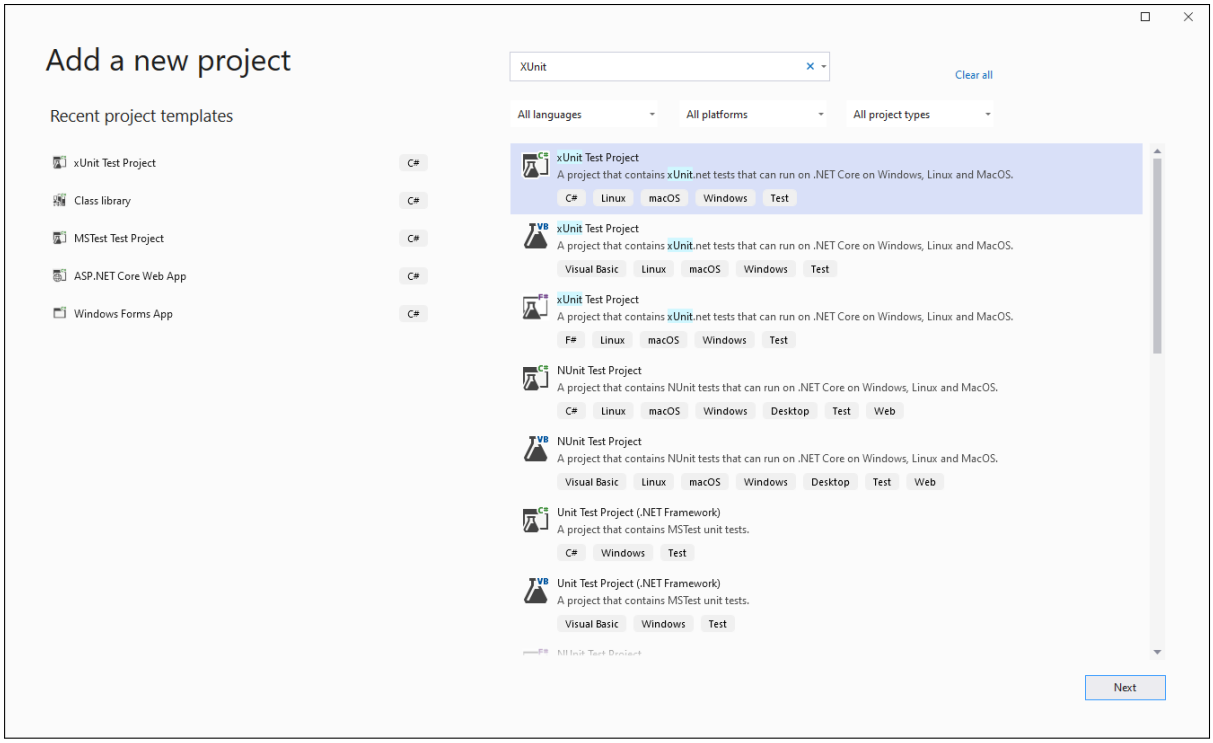

Figure 3: Új XUnit projekt létrehozása

Különbségek az `MSTest`-hez képest:

1. Míg az MSTest esetén a tesztosztályból egyetlen példány készül egy csoportos teszt futtatás esetén, úgy az XUnit esetén minden tesztesethez külön objektum kerül példányosításra a teszt osztályból, azaz itt az osztály konstruktorra minden egyes teszteset futása előtt meghívódik.
2. Az egyes tesztesetek utáni "takarítás" az `IDisposable` interfész megvalósításával a `Dispose()` függvény segítségével lehetséges.
3. Az egyes teszt metódusok a `[Fact]` annotációval kerülnek ellátásra, paraméterezett tesztesetek a `[Theory]` attribútummal definiálhatóak.
4. Az `XUnit.Assert` osztály más néven tartalmazza a vizsgálathoz használt függvényeket (pl. `IsEqual` helyett `Equal`, `IsTrue` helyett `True`), illetve elérhetők külön vizsgálatok az egy elemű, illetve üres listák vizsgálatára (`Assert.Empty` és `Assert.Single`).