Eseményvezérelt alkalmazások: 5. gyakorlat

A munkafüzetben megismerkedünk a modell réteg egységtesztelésének menetével. Többféle, .NET környezetben használható tesztelési keretrendszert kipróbálunk, illetve a perzisztencia réteg *mockolásával* is megismerkedünk.

Szavak számlálásának tesztelése

Készítsünk egységtesztelést a 4. gyakorlat során továbbfejlesztett `DocuStat` projekt `DocumentStatistics` osztályához.

Tesztelés megvalósítása MSTest használatával

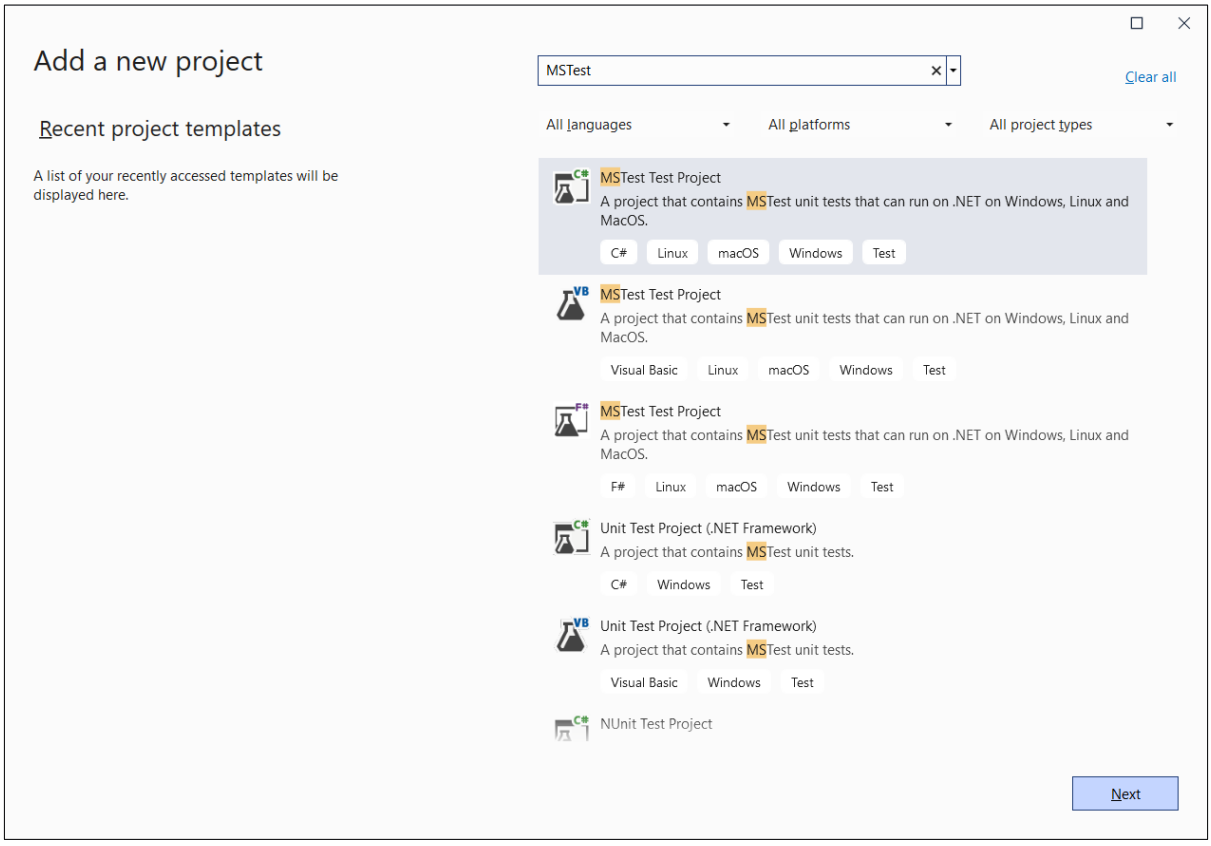

Figure 1: Új MSTest projekt létrehozása

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 és az új projektnek adjuk a `DocuStatTest` nevet.

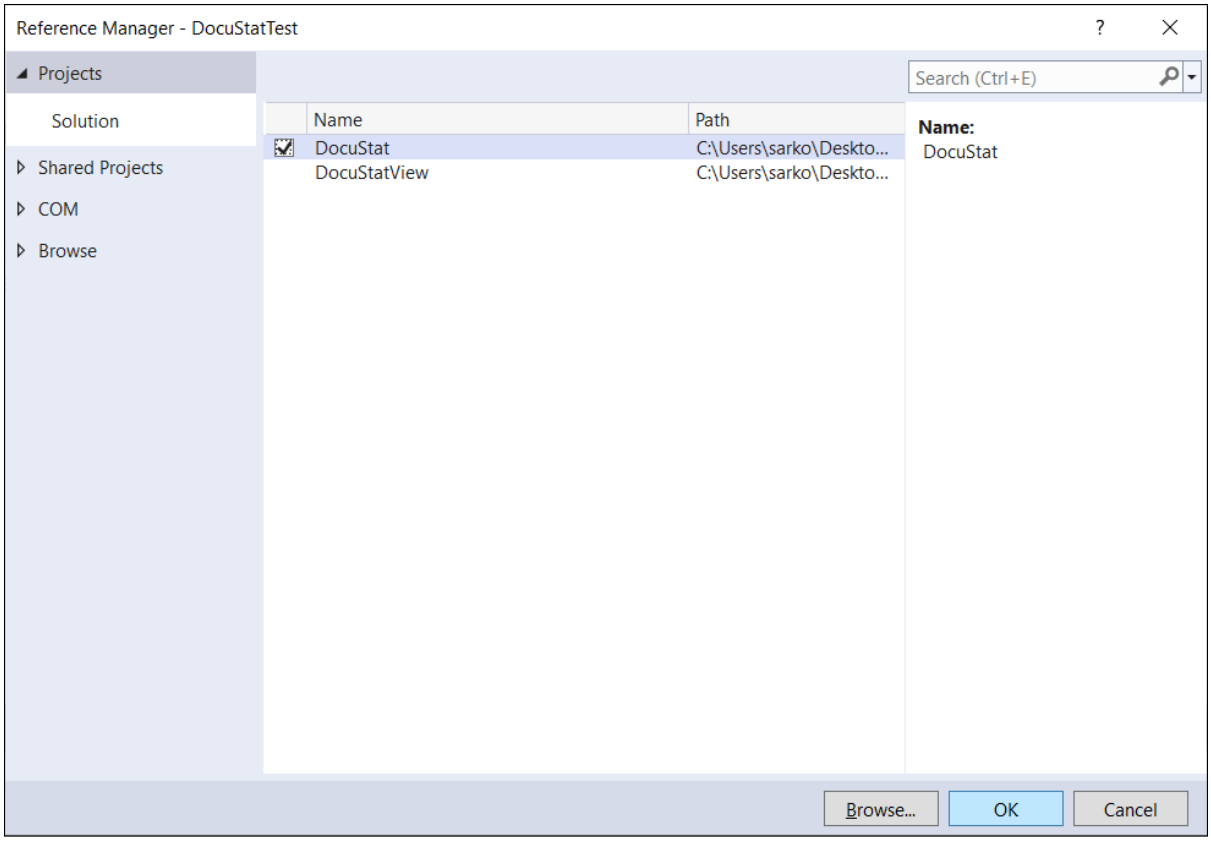

Figure 2: Projekt referencia hozzáadása

A létrejött projektben hivatkozzunk a `DocuStat`-ra: jobb klikk a `DocuStatTest`-re és *Add -> Project Reference*.

A létrejött osztályt (és fájlt) nevezzük át `DocumentStatisticsTest`-re és valósítsuk meg a `DocumentStatistics` osztály tesztelését benne.

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.

A `DocumentStatistics` osztály 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 könyvtár hozzáadásához jobb klikk a `DocuStatTest` projektre *Manage NuGet Packages* és az előugró ablakban a *Browse* tabon keressünk rá csomag nevére és telepítsük.

Ahhoz, hogy ellenőrizzük, hogy a tesztelt 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, hogy 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 `DocumentStatistics` osztály példányosításakor az `IFileManager` típusú paraméter legyen egy, a `Moq` package segítségével példányosított osztály.

```
private Mock<IFileManager> _mock = null!;
private DocumentStatistics _docStats = null!;

[TestInitialize]
public void InitDocuStatTest()
{
    _mock = new Mock<IFileManager>();
    _docStats = new DocumentStatistics(_mock.Object);
}
```

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

Például:

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

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

Fájl betöltés tesztelése ^{KM}

Az `IFileManager` interfész `Load` függvényének mockolásával teszteljük a `DocumentStatistics` osztály `Load` függvényét.

Valósítsuk meg az alábbi eseteket:

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.
3. Ellenőrizzük a modell `FileContentReady` és `TextStatisticsReady` eseményeinek kiváltását is.

Amennyiben az a helyes viselkedés, hogy egy tesztfüggvény kivétellel zárul, akkor ezt jelezhetjük az `[ExpectedException]` annotációval.

Szó számláló tesztelése ^{EM}

Miután meggyőződtünk róla, hogy a `Load` függvényünk jól működik, ellenőrizzük a `DistinctWordCount` propertyt is a `FileContent` ellenőrzésének mintájára.

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.

Megjegyzés: Annak érdekében, hogy biztosítsuk azt, hogy a mellékhatásos tesztesetek ne legyenek hatással egymásra, megvalósíthatunk egy `[TestCleanUp]` annotációval ellátott metódust, amiben a szükséges értékek visszaállítását megtehetjük. 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.

Egyéb számlálók tesztelése ^{EM}

Ellenőrizzük a karakterszámlálás működését (`CharacterCount`):

1. Amennyiben egy általános szöveget adunk meg, akkor a szöveg hosszát kapjuk-e?
2. Üres bemenet esetén a megfelelő értéket adja-e?

Ellenőrizzük a nem whitespace karakterszámlálás működését (`NonWhiteSpaceCharacterCount`):

1. Amennyiben egy általános szöveget adunk meg, akkor a jó eredményt kapjuk-e?
2. Csak whitespace bemenet esetén a megfelelő értéket adja-e?

Ellenőrizzük a mondatszámlálás működését (`SentenceCount`):

1. Amennyiben egy általános szöveget adunk meg, akkor a jó eredményt kapjuk-e?
2. Üres bemenet esetén a megfelelő értéket adja-e?

Ellenőrizzük a tulajdonnevek számlálását (`ProperNounCount`):

1. Amennyiben szerepelnek nagybetűs szavak egy szöveg közepén, akkor megfelelő eredményt kapunk-e?
2. Több mondat esetén a megfelelő eredményt kapjuk-e?

További tesztesetek ^{OP}

ColemanLieuIndex számításának tesztelése:

1. Egy többmondatos szövegre a megfelelő eredményt kapjuk-e?
2. Egy mondat esetén jó eredményt kapunk-e?

FleschReadingEase számításának tesztelése:

1. Egy többmondatos szövegre a megfelelő eredményt kapjuk-e?
2. Magánhangzó nélküli szöveg esetén megfelelő eredményt kapunk-e?

Tesztelés megvalósítása xUnit használatával

Ismerkedjünk meg egy újabb, `xUnit` névre hallgató tesztelési keretrendszerrel is. Adjunk a `Solution`-hoz egy újabb projektet `DocuStatTestXUnit` néven.

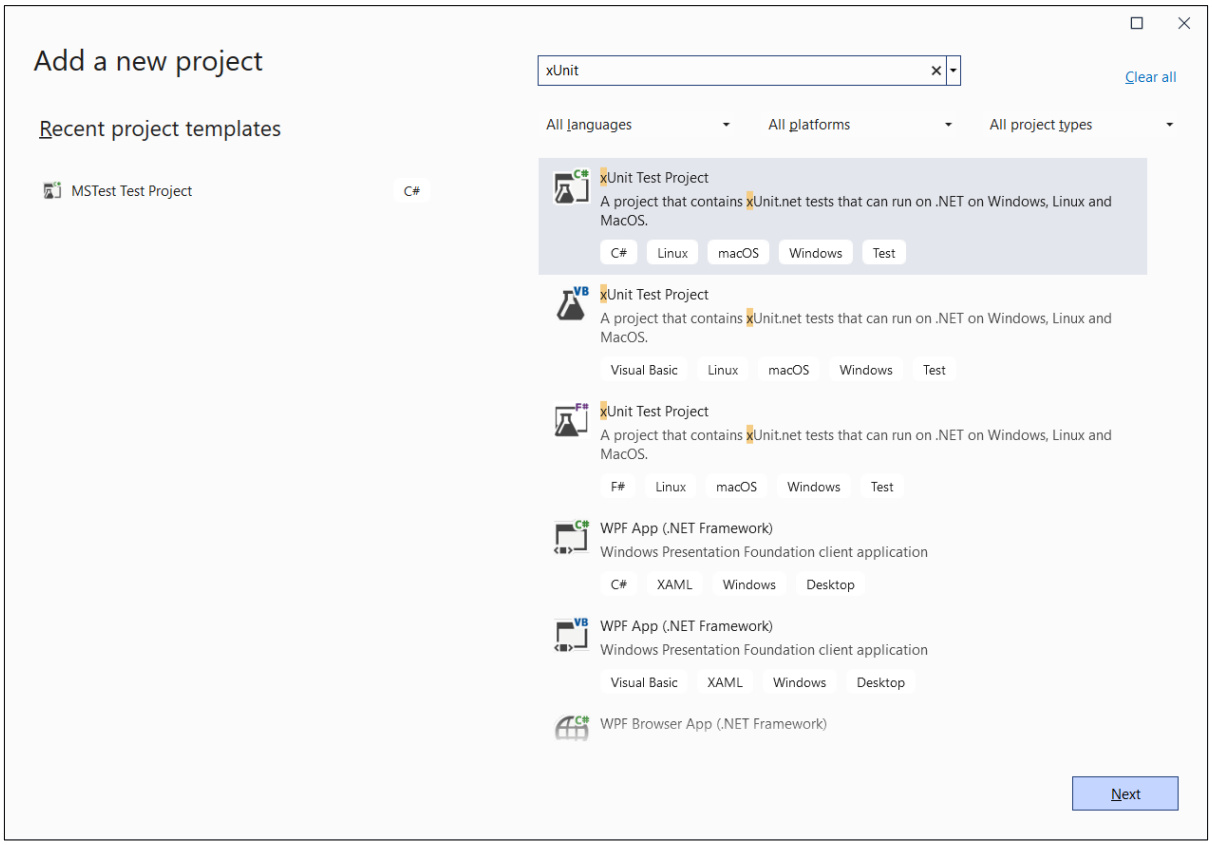

Figure 3: Új XUnit projekt létrehozása

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`).

Tesztesetek megvalósítása ^{EM}

Az MSTest keretrendszerrel elkészített teszteket valósítsuk meg az `xUnit` segítségével is.