**Toegepaste Informatica**

**Opdracht Software Testing**

*Testen in .Net omgeving*

**Jan Dammans**

(r0796935)

Bachelor – Toegepaste Informatica

**Software Testing**

Frank Serneels

Academiejaar 2021-2022

Inhoud

[1 Week 1 - Onderzoek 4](#_Toc88727622)

[1.1 Tools 4](#_Toc88727623)

[1.2 Verschil xUnit – Nunit 4](#_Toc88727624)

[1.3 Cursus 4](#_Toc88727625)

[1.4 Bronnen 4](#_Toc88727626)

[2 Week 2 – Basis principes (theorie) 5](#_Toc88727627)

[2.1 Doel van testen 5](#_Toc88727628)

[2.2 Testfase 5](#_Toc88727629)

[2.3 Soorten automatische test 5](#_Toc88727630)

[2.4 TestPrincipe 5](#_Toc88727631)

[2.5 3A stappenplan 6](#_Toc88727632)

[3 Week 3 -Basistesten schrijven + theorie 7](#_Toc88727633)

[3.1 Simpele test 7](#_Toc88727634)

[3.1.1 Calculator 7](#_Toc88727635)

[3.1.2 Som van 2 elementen 7](#_Toc88727636)

[3.1.3 Som van meerdere element 8](#_Toc88727637)

[3.1.4 Resultaat 8](#_Toc88727638)

[3.1.5 Vermenigvuldigen 9](#_Toc88727639)

[3.1.6 Delen 10](#_Toc88727640)

[3.1.7 Resultaat 10](#_Toc88727641)

[4 Week 4 – Uitbreiding basiskennis (Exceptions, mocktesten) 11](#_Toc88727642)

[4.1 Lifecycle unit testen 11](#_Toc88727643)

[4.1.1 Resultaat 11](#_Toc88727644)

[4.2 Exceptions 12](#_Toc88727645)

[4.2.1 Testen en uit uitleg 12](#_Toc88727646)

[4.2.2 Soorten exceptions 16](#_Toc88727647)

[5 Week 5 - Mocktesten 17](#_Toc88727648)

[5.1 Mock testen 17](#_Toc88727649)

[5.1.1 Stappenplan 17](#_Toc88727650)

[5.1.2 Voorbeeld 18](#_Toc88727651)

[6 Week 6 – Integration tests 20](#_Toc88727652)

[6.1 Doel intagration tests 20](#_Toc88727653)

[6.2 Uitwerken 20](#_Toc88727654)

[6.2.1 Controller 20](#_Toc88727655)

[6.2.2 Klasse 21](#_Toc88727656)

[6.2.3 Testserver 21](#_Toc88727657)

[6.2.4 Database 22](#_Toc88727658)

[6.2.5 Testen 23](#_Toc88727659)

[7 Week 7 – Integration tests 25](#_Toc88727660)

[7.1 Apply integration testing to web services 25](#_Toc88727661)

[7.2 Testing Web Api Controllers 25](#_Toc88727662)

[8 Week 8 – Integration tests (vervolg) 26](#_Toc88727663)

[8.1 Integratietesten schrijven 26](#_Toc88727664)

[8.2 Fluent Assertions 28](#_Toc88727665)

[9 Week 9 – Finale test 30](#_Toc88727666)

[10 Logboek 31](#_Toc88727667)

[11 Ervaring 33](#_Toc88727668)

# Week 1 - Onderzoek

## Tools

* Visual Studio
* C#
* Selenium
* xUnit

## Verschil xUnit – Nunit

1. xUnit beter voor nieuwe/lege projecten
2. xUnit propere/duidelijkere versie van NUnit
3. xUnit makkelijk uitbreid baar (zoals Xamarin)
4. xUnit is nieuwer dus veranderd soms nog
5. xUnit heeft beter geïsoleerde tests omdat iedere test wordt geïnstantieerd. Zo kunnen de testen elkaar niet beïnvloeden.
6. NUnit kunnen meerdere keren testen hetzelfde benoemen zoals SetUp wat voor complexiteit zorgt.
7. xUnit heeft duidelijkere namen zoals Assert.throw ipv ExpectedExceptation

## Cursus

1. <https://openclassrooms.com/en/courses/5670586-test-your-net-applications>

## Bronnen

1. <https://www.lambdatest.com/blog/xunit-testing-tutorial/>
2. <https://www.lambdatest.com/blog/setting-selenium-webdriver-for-xunit/>
3. <https://code-maze.com/automatic-ui-testing-selenium-asp-net-core-mvc/>
4. Boek: programmeren in c#: hoofdstuk testen

# Week 2 – Basis principes (theorie)

## Doel van testen

Fouten voorkomen die op hun beurt veel geld zouden kosten.

## Testfase

* Reactief: bugs vinden nadat ze zijn opgetreden
* Proactief: bugs voorkomen door te testen

Soorten testen:

* Manueel
  + Zelf gaan kijken of alles klopt
* Automatisch
  + Alles automatisch laten verlopen
  + Soorten automatische testen:
    - **Unit test**: Deze testen een deel van de code die geïsoleerd is van alle afhankelijkheden. Deze zijn eenvoudig te ontwikkelen, kunnen snel worden uitgevoerd, en stellen je in staat om kleine delen van je code direct te controleren.
    - **Mock tests** - Dit zijn unit tests die een "mock" van een deel van de code testen in plaats van het werkelijke deel van de code zelf.
    - **Integratie tests** - Deze controleren of al deze secties van geïsoleerde code correct samenwerken. Het duurt over het algemeen langer dan unit tests om ze uit te voeren en ze kunnen afhankelijkheden testen waar de ontwikkelaar controle over heeft, zoals databases of bestanden.

## Soorten automatische test

**Test-Driven development**

* Eerst testcases, dan code
* Blijven testen totdat alles werkt (proactief – vooral technisch)

**Behaviour-Driven Development**

* Verbeterd communicatie alle teams
* Vooral klantgericht (vooral output)

## TestPrincipe

**F**ast,

**I**solated,

**R**epeatable,

**S**elf-valifating,

**T**imely

## 3A stappenplan

* **Arrange:** Alle noodzakelijke randvoorwaarden en inputs voor het te testen object of de te testen methode regelen.
* **Act:** Voer de actie uit op het geteste object of de geteste methode met de gerangschikte parameters.
* **Assert:** Bevestigen (of verifiëren) dat de actie van het geteste object of de geteste methode zich gedraagt zoals verwacht.

# Week 3 -Basistesten schrijven + theorie

## Simpele test

Eerst wordt de methode uitgewerkt in code (zie Calculator). Vervolgens maken we dan twee tests die nagaan of de code in de method werkt.

### Calculator

Om te beginnen hebben we een methode die een array krijgt met nummers. Vervolgens worden deze getallen allemaal bij elkaar opgeteld. Het resultaat returnen we vervolgens.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Som van 2 elementen

Vervolgens maken we onze test klasse waarin nagegaan wordt of de test al dan niet lukt.

Eerst wordt er een nieuwe calculator aangemaakt. Vervolgens roepen we de methode op die de getallen optelt. Het resultaat houden we bij zodat we dit kunnen vergelijken met het resultaat dat we verwachten.

Tot slot vergelijken we het te verwachten resultaat met het effectieve resultaat. In dit geval verwachten we dat 1 + 1 gelijk is aan 2. Om dit aan te geven maken we gebruik van de Assert.Equal.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Som van meerdere element

Vervolgens gaan we ook meerdere getallen optellen en controleren of dit werkt. Het enige dat we hierbij aanpassen, zijn de variabelen die we meegeven aan de methode Sum.

Uiteraard moeten we dan ook het verwachtte resultaat aanpassen. Bijkomend willen we ook nagaan of het type dat we terugkrijgen ook voldoet aan de voorwaarden. Dit doen we met de Assert.IsType.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Resultaat

Bij het runnen van de tests zien we dat alles werkt. Bijgevolg kan de method Sum() in productie gebracht worden omdat we weten dat alles correct werkt.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Vermenigvuldigen

Vervolgens werken we vermenigvuldigen net op dezelfde manier uit als het optellen. Het verschil hierbij is dat de berekening anders verloopt.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Delen

Ook het delen zal op dezelfde manier verlopen als het optellen. Uiteraard delen we de getalen.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Resultaat

Tot slot runnen we de tests. Hierbij zien we dat alle tests geslaagd zijn. We kunnen dus de methods van de calculator in productie brengen.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

# Week 4 – Uitbreiding basiskennis (Exceptions, mocktesten)

## Lifecycle unit testen

Meerdere componenten testen is eigenlijk een verzameling van subcomponenten testen. Hiervoor hebben twee elementen nodig:

1. Constructor
2. IDisposable

De constructor dient om de klasse te kunnen instantiëren en de interface IDisposable geeft aan dat je klasse middelen moet vrijmaken wanneer ze niet langer in gebruik zijn, en de methode Dispose wordt beschikbaar gesteld zodat gebruikers van je klasse deze kunnen aanroepen om de verbruikte middelen vrij te maken

Als eerste restten we de rekenmachine om dan vervolgens 2 getallen op te tellen. Tot slot vergelijken we of de methode die 7 + 3 moet uitvoeren weldegelijk 10 is.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

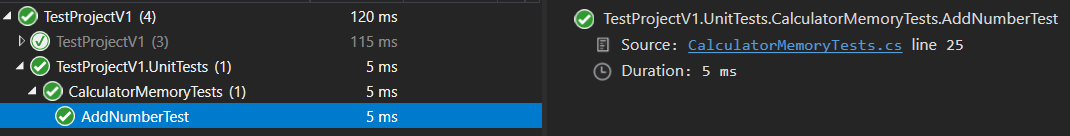
Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Resultaat

Wanneer we met een stopwatch werken, wordt de volgorde van bewerking getoond. Dit is zeer interessant omdat we op die manier een beter inzicht krijgen.

We zien dat de test geslaagd is en bijgevolg kan de methode in productie gebracht worden.



## Exceptions

Het is belangrijk om exceptions te throwen omdat je op die manier veel meer controle hebt over je code. Dit kan je doen door gebruik te maken van de Assert.Throws functie.

### Testen en uit uitleg

#### DivideByZeroException

Om het concreter te maken, je kan bijvoorbeeld niet delen door 0. Bijgevolg zal er dus een fout optreden waardoor je programma crasht. Aangezien je weet dat er een fout zal optreden, kan je dit opvangen.

Hiervoor schrijven we de code uit:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Vervolgens runnen we deze en zoals verwacht lukt de test niet.

Afbeelding met tekst, schermafbeelding, monitor, scherm

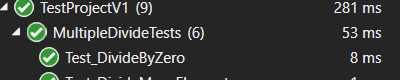
Automatisch gegenereerde beschrijving

Dit komt omdat het type geen bool is maar een error. We kunnen dit opvangen door errond een throws exception te geven. Dit geeft aan dat we weten dat er een error zal optreden. Enkel indien er dan een dividebyzeroexception optreedt, zal de test geslaagd zijn.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Zoals verwacht slaagt de test nu.



Om aan te tonen dat de test enkel slaagt indien er een error optreedt veranderen we 8/0 door 8/1. Er zal bijgevolg geen fout optreden omdat dit geheel juist is.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We zien dat de test niet slaagt en dat komt omdat we aangeven dat we een error verwachten terwijl dit niet zo is.

Afbeelding met tekst, schermafbeelding, monitor

Automatisch gegenereerde beschrijving

#### InvalidOperationException

Een ander voorbeeld is een invalid operation exception. Hiervoor schrijven we eerst een methode die valideert of de waarde al dan niet geldig is. Indien de waarde niet geldig is, wordt er een invalid exception error gethrowd.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Bij het vermenigvuldigen geven we 2 keer maximale waarde mee aan de validatie

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Zoals verwacht lukt de test niet omdat er bij de validatie een throw wordt gedaan.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Wat we nu nog moeten doen is aangeven dat we een invalid operation error verwachten.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Zoals we hebben voorzien, slaagt deze test.Afbeelding met tekst, schermafbeelding, monitor, zwart

Automatisch gegenereerde beschrijving

### Soorten exceptions

Hieronder is er een overzicht met alle soorten exceptions met de uitleg wanneer ze optreden.

| Exception Class | Description |
| --- | --- |
| [ArgumentException](https://docs.microsoft.com/en-us/dotnet/api/system.argumentexception?view=netframework-4.8) | Raised when a non-null argument that is passed to a method is invalid. |
| [ArgumentNullException](https://docs.microsoft.com/en-us/dotnet/api/system.argumentnullexception?view=netframework-4.8) | Raised when null argument is passed to a method. |
| [ArgumentOutOfRangeException](https://docs.microsoft.com/en-us/dotnet/api/system.argumentoutofrangeexception?view=netframework-4.8) | Raised when the value of an argument is outside the range of valid values. |
| [DivideByZeroException](https://docs.microsoft.com/en-us/dotnet/api/system.dividebyzeroexception?view=netframework-4.8) | Raised when an integer value is divide by zero. |
| [FileNotFoundException](https://docs.microsoft.com/en-us/dotnet/api/system.io.filenotfoundexception?view=netframework-4.8) | Raised when a physical file does not exist at the specified location. |
| [FormatException](https://docs.microsoft.com/en-us/dotnet/api/system.formatexception?view=netframework-4.8) | Raised when a value is not in an appropriate format to be converted from a string by a conversion method such as Parse. |
| [IndexOutOfRangeException](https://docs.microsoft.com/en-us/dotnet/api/system.indexoutofrangeexception?view=netframework-4.8) | Raised when an array index is outside the lower or upper bounds of an array or collection. |
| [InvalidOperationException](https://docs.microsoft.com/en-us/dotnet/api/system.invalidoperationexception?view=netframework-4.8) | Raised when a method call is invalid in an object's current state. |
| [KeyNotFoundException](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.keynotfoundexception?view=netframework-4.8) | Raised when the specified key for accessing a member in a collection is not exists. |
| [NotSupportedException](https://docs.microsoft.com/en-us/dotnet/api/system.notsupportedexception?view=netframework-4.8) | Raised when a method or operation is not supported. |
| [NullReferenceException](https://docs.microsoft.com/en-us/dotnet/api/system.nullreferenceexception?view=netframework-4.8) | Raised when program access members of null object. |
| [OverflowException](https://docs.microsoft.com/en-us/dotnet/api/system.overflowexception?view=netframework-4.8) | Raised when an arithmetic, casting, or conversion operation results in an overflow. |
| [OutOfMemoryException](https://docs.microsoft.com/en-us/dotnet/api/system.outofmemoryexception?view=netframework-4.8) | Raised when a program does not get enough memory to execute the code. |
| [StackOverflowException](https://docs.microsoft.com/en-us/dotnet/api/system.stackoverflowexception?view=netframework-4.8) | Raised when a stack in memory overflows. |
| [TimeoutException](https://docs.microsoft.com/en-us/dotnet/api/system.timeoutexception?view=netframework-4.8) | The time interval allotted to an operation has expired. |

# Week 5 - Mocktesten

## Mock testen

Mock testen zorgt er voor dat waarden die worden aangeleverd kan override worden in de specifieke test. Zo kan je een resultaat van Datetime.Now() veranderen door een zelf gekozen datum.

In de meeste gevallen wordt dit gebruikt om hele omgevingen te gaan simuleren. Denk maar aan moment waarop de applicatie veel respons heeft, tijdswisselingen, andere tijdstippen. Al snel werd duidelijk dat mock-testen veel vertellen over het AS-IS maar kan tevens gebruikt worden om sneller reallife fouten te achterhalen die je met gewone testdata niet zo snel oplost.

### Stappenplan

Om mock data te implementeren, volg je best deze stappen:

1. Identificeer en verduidelijk de mocks die wilt gebruiken
2. Boots het gedrag na van je componenten
3. Bevestig je mock code

### Voorbeeld

De bedoeling is dat we er voor zorgen dat Datetime.now() niet het huidige tijdstip weergeeft maar wel een tijdstip die we willen mocken naar een ander moment.

We willen er voor zorgen dat op 1 januari 2100 er een melding wordt gegeven dat de server onderhouden moet worden. Hiervoor maken we gebruik van een invalid operation exception.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Vervolgens maken we een interface aan en de classe zelf die er voor zorgen dat het huidige tijdstip wordt weergegeven.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Om mockdata te kunnen gebruiken, moeten we eerst de NuGet-package “Moq” installeren.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Daarna kunnen we de code aanpassen zodat we kunnen doen alsof het nu 1 januari 2100 is.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Wanneer we de test runnen, zien we dat de test niet slaagt.

Afbeelding met tekst, schermafbeelding, monitor, scherm

Automatisch gegenereerde beschrijving

Dat komt omdat de invalid operation error is gethrowd. Op zich werkt het dus maar we willen juist dat de test slaagt. Vandaar dat we deze error moeten opvangen

Afbeelding met tekst, schermafbeelding, scherm, zilver

Automatisch gegenereerde beschrijving

Zoals verwacht is deze test nu geslaagd.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

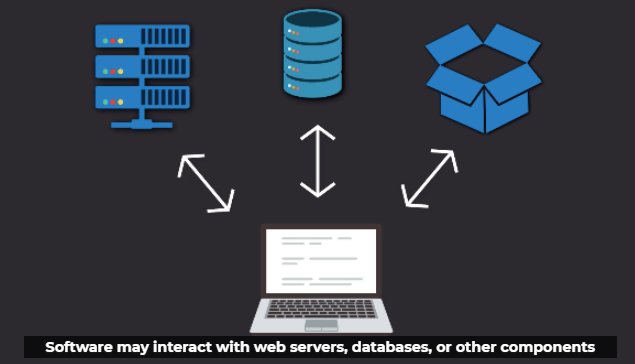
# Week 6 – Integration tests

## Doel intagration tests

Het handmatig testen van.NET applicaties is moeilijk en neemt veel tijd in beslag voor één persoon. Zelfs geautomatiseerde unit testen voldoen niet aan de volledige behoeften. Unit tests zijn over het algemeen bedoeld voor het testen van specifieke componenten in geïsoleerde testomgevingen.

Software daarentegen heeft niet de neiging om in geïsoleerde omgevingen te werken. Het wordt gebruikt over verschillende componenten heen.

Een integratietest evalueert de individuele testen in een afgebakende omgeving met zijn eigen set testen. Door na te gaan dat de verschillende delen van de applicatie correct werken wanneer ze gegroepeerd zijn, kan de manier waarop sz applicatie test op een bredere schaal geautomatiseerd worden.



## Uitwerken

Als voorbeeld gebruik ik een ASP .Net Core Web Api applicatie.

### Controller

In de controller voegen we de route toe die aangeeft dat bij het oproepen van de route Sum de methode wordt opgeroepen om de som te bereken.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Klasse

Vervolgens maken we de klasse Calculator aan die de som zal berekenen.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Testserver

Testen tegen controllers en MVC in ASP.NET helpt om je code georganiseerd te houden en volgens een zeer testbaar patroon. Een **test** **host** is een set van testen en specifieke componenten die u in staat stellen om te interageren met uw Web API en server op een test-gedreven manier en met productie-georiënteerd gedrag. Door de unit tests te scheiden van de integratie tests in verschillende projecten, kun je ervoor zorgen dat componenten in uw infrastructuur granulair getest kunnen worden.

Vervolgens creëren we een test context waarin we de testserver zullen maken.

Om te beginnen hebben we een TestServer en een HttpClient nodig. Vervolgens is er dan de methode die de server zal opstarten. Deze schrijven we uit en krijgen volgende code.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Database

Omdat in quasi elke applicatie wel een database zit, moeten er ook integratietesten uitgevoerd worden op databases omdat deze met meerdere communiceren. In een .Net omgeving is dit zeer vaak Entitiy Framework die achter de databases zit. Een van de voordelen van Entity Framework is de optie van een in-memory database die in elke test kan worden gestart en verwijderd. Ook kan er hierin data worden gezet en bewerkt die na de tests eenvoudig verwijderd kunnen worden.

Als eerste moeten we de InMemory NuGet package installeren.

Vervolgens hebben we een DbContext die we moeten creëren zodat we een database kunnen aanmaken.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Entiry Framework is zeer testvriendelijk. Zo kan je gewoon je testklasse schrijven in de context zelf.

Vervolgens maken we een service die zal zorgen voor het schrijven van en naar de database. We voorzien een methode die een payment kan creëren. De waarde die we krijgen vanuit de repository return we.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Hierin maken we ook een Repository aan. De repository schrijft en leest de data van de databank die dan vervolgens terug aan de service. Als test returnen we nu true omdat dit geen invloed heeft op de effectieve werking.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

### Testen

Tot slot maken we een nieuw project aan met daarin een test dat nagaat of de database werkt.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Wanneer we de test runnen, zien we dat de test geslaagd is en we dus kunnen concluderen dat het schrijven van de CreatePayment method werkt.

Afbeelding met tekst, schermafbeelding, monitor

Automatisch gegenereerde beschrijving

Om te illustreren wanneer een operatie faalt, veranderen we de return value van in de repository van “true” naar “false”.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Zoals verwacht faalt de test en kunnen we concluderen dat we nu kunnen nagaan of een operatie al dan niet geslaagd is.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

# Week 7 – Integration tests

## Apply integration testing to web services

## Testing Web Api Controllers

**Nota:** Voor dit stuk heb ik een andere mini-cursus geraadpleegd om wat te kunnen veriepen namelijk: <https://dotnettutorials.net/lesson/kestrel-web-server-asp-net-core/> .

Integratietesten zorgen er voor dat je specifieke delen kunt testen of ze wel degelijk samenwerken zoals het moet. In tegenstelling tot unit tests, hebben integratietests vaak te maken met de infrastructuur van de applicatie.

Een default ASP.NET Core server is gebasseerd op het Kestrel cross-platform server. Deze heeft volgende kenmerken:

* Snel, licht en asynchrone input/output processing
* Ondersteuning van alle platformen inclusief versies dat .NET ondersteunt
* Eenvoudig te implmenteren in de API code

Het grootste voordeel is dat al het bovenstaande bij de integratietesten automatisch al verweven zitten omdat he gebruik maakt van de test-server.

# Week 8 – Integration tests (vervolg)

## Integratietesten schrijven

Integratietesten zijn over het algemeen bedoeld om te verzekeren dat een bepaalde URL in je webserver bereikbaar is. Het schrijven van integratietesten maakt het volgende mogelijk:

* Een HTTP-gebaseerde aanroep kan overal worden vanuit elke ondersteunde client aangeroepen worden (Bv. bij de applicatie als eindgebruiker, vanuit een mobiele app of webpagina).
* De Web API die de inhoud weergeeft, kan worden aangesproken worden naar een specifieke URL, zodat de inhoud van het verzoek effectief de HTML is zoals er verwacht wordt.
* Inhoud in de respons komt zoals verwacht (meestal in JSON formaat, maar andere inhoudstypes worden ook ondersteund), met het resultaat van de operatie die de client heeft aangevraagd.

Eerst moeten er een controller gemaakt worden die aangesproken zal worden. Hierbij definiëren we de route die we zullen oproepen. Voor de controller is dit api/Calculator vanwege de controllernaam. Vervolgens is er een Sum methode. Dit is een GET-methode en het pad is vervolgens Sum. In de deze methode maken we de som.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We voegen de methode toe die zal testen of de uitgevoerde api call werkt.

Om te beginnen voeren we een Get call met als parameters x=1 en y=2. Vervolgens testen we of er weldegelijk een succesvolle code is gereturned. Dit kan je ook met onder andere foutcodes.

Vervolgens wordt de inhoud gelezen van de gekregen api call. Nadien controlleren we of de som 1 + 2 wel degelijk 3 geeft via de Assert.true met true en dan de inhoud van api call.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Bij het runnen van deze tests, zien we dat de testen geslaagd zijn en dat er zowel een succesvolle call wordt gemaakt alsook het juiste resultaat.

Afbeelding met tekst, schermafbeelding, monitor, buiten

Automatisch gegenereerde beschrijving

Om te testen of de test weldegelijk klopt, veranderen we enkel de twee parameters door 2 en 3. Dit resulteert in 5 dus zou de test nu moeten falen.



Zoals verwacht faalt de test en kunnen we concluderen dat de test op een juiste manier werkt.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

## Fluent Assertions

Fluent Assertions zijn een set van .NET extensie methodes die toelaten om .Asserts uit te voeren op de outputs van Web API en HTTP-gerelateerde content en resultaten. Ze kunnen worden gecombineerd met zowel TDD als BDD benaderingen. Dit maakt het een zeer sterke uitvoering.

Om Fluent Assertions in je project te hebben, moet eerst de NuGet package Fluent Assertion classes beschikbaar te maken in je project.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Vervolgens implementeren we de Fluent Assertions. In plaats van manueel te gaan controleren of de inhoud van die bepaalde context komt met de inhoud kunnen we nu bijvoorbeeld van de methode .Should().Be() gebruik maken.

We vervangen de Assert.Equal door de Should be methode.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Wanneer we dit runnen zien we dat de testen slagen.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Om na te gaan dat de test consistent is vervangen we de parameters door 2 en 3

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We zien dat de test faalt, zoals we verwachten, maar ook dat de excpected en actual vervangen wordt door een message wat het wel net iets minder duidelijk maakt.

# Week 9 – Finale test

Deze week heb ik de finale test van de course doorlopen. Hieronder zie je het behaalde resultaat.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

# Logboek

|  |  |  |
| --- | --- | --- |
| **Datum** | **Wat** | **Duur** |
| 20/09/2021 - 26/09/2021 | Onderzoek verrichten naar alle mogelijkheden van C# inclusief voor- en nadelen | 2 uur 30 minuten |
| 27/09/2021 - 03/10/2021 | * Theorie xUnit testen * Waarom is testen belangrijk * Voordelen testen * Doel van testen * Soorten testen * Unit * Mock * Integration test * Basis testen uitvoeren | 4 uur |
| 04/10/2021 - 10/10/2021 | * Kennis xUnit uitbreiden * Intro complexere xUnit testen * intro * Test-drive development * Test cases * TDD lifecycle * Behaviour-driven development * Gherkin * Cucumber * Verschil BDD en TDD * Wanneer BDD en TDD * Unit test theorie * Eerste 2 unit test volgens F.I.R.S.T principe * Basis exceptions | 4 uur 30 minuten |
| 11/10/2021 - 18/10/2021 | * 3 AA principle * Theorie Assert * Unit test lifecycle * Uitbreiden tests * Verdiepen MSDN documentatie van exceptions in unit testing * Theorie mocktesten * Opstarten mocktesten | 4 uur 20 minuten |
| 18/10/2021 -24/10/2021 | * Dieper ingaan op exceptions * Soorten exceptions * Mocktesten voorbeelden * Mocktesten verder uitwerken | 2 uur 30 minuten |
| 25/10/2021 – 31/10/2021 | * Theorie intregation tests * Opstarten voorbeeld integration test * Database en integratietesten | 2 uur |
| 01/11/2021 – 07/11/2021 | Herfstvakantie - Rust genomen | / |
| 08/11/2021 – 14/11/2021 | * Kennis Testing Web API vergaren * Andere cursus geraadpleegd om te verdiepen in Kestrel: https://dotnettutorials.net/lesson/kestrel-web-server-asp-net-core/ | 3 uur |
| 15/11/2021 –  21/11/2021 | * Integratie testen schrijven * Testen of api wel degelijk is opgeroepen op basis van response code * Testen of juiste waarde in api call zit * Fluent Assertions | 5 uur 45 minuten |
| 22/11/2021 –  28/11/2021 | * Finale test * Certificaat behaald | 1 uur |

# Ervaring

De eerste uren in de cursus waren zeer leerrijk. Ondanks we wel weten dat testen belangrijk is, werd er toch nog enkele voorbeelden gegeven waar het in de realiteit misgaat. Op die manier weet je dus waar de valkuilen zich bevinden en op welke manier je testen maken moet aanpakken.

Vervolgens waren er de testen zelf. Bij iedere test hoorde wel een redelijk blok theorie. Zo is er bij bijvoorbeeld Exception handeling een heel wat uitgebreider stuk theorie aan gekoppeld dan oorspronkelijk gedacht.

Persoonlijk dacht ik dat mocktesten met mockdata niet echt nuttig waren en dat er beter getest kan worden met de huidige data. Wat ik dankzij de cursus heb ingezien is dat je door gebruik te maken van mockdata je je kan verplaatsen naar een virtueel moment. Als je nu iets zou testen zal dat waarschijnlijk niets speciaals opleveren maar wat wanneer de applicatie bijvoorbeeld runt terwijl er een jaarovergang is. Via mockdata kan je dus het huidig tijdstip veranderen naar een testen moment.

Duidelijke namen geven aan de testen is belangrijker dan ik dacht. Op die manier zie je duidelijker wat heb je al getest en wat zou je mogelijks nog kunnen testen. Soms merkte ik ook dat een test bijvoorbeeld te weinig inhoud bevatte. Zo kon je bijvoorbeeld enkel vermenigvuldigen met 2 getallen. Zo kwam ik snel tot de vaststelling dat je beter meer testen schrijft.

Entity Framework is zeer testvriendelijk. Zo kan je zelf kiezen of je de testen bij de klasse zelf schrijft of je deze in het algemene testproject zet. Mijn voorkeur gaat uit om dit te houden in het testproject zodat je consistent blijft. Indien je enkel testen schrijft voor de database kan je dit wel in de klasse zelf doen.

In tegenstelling tot unit tests, hebben integratietests vaak te maken met de infrastructuur van de applicatie. Intregatietests zijn dus veel ruimer, belangrijker en hebben dus een grotere impact op de werking van de applicatie.

Een standaard ASP.NET Core server is gebaseerd op Kestrel, dat een cross-platform webserver is. Hiervan had ik eerder nog nooit gehoord maar dit blijkt een uiterst interessante webserver te zijn.

Fluent Assertions maken het mogelijk om zowel compactere als leesbaardere code te schrijven en te laten uitvoeren. Een nadeel is wel dat de actual en expected vervangen wordt door een message wat het onnodige onduidelijk maakt bij kleine testjes.