

FUNKTIONELL INTEGRATIONSTESTNING AV ETT WEBB-API

Jens Henriksson



2021:34

Datum för godkännande: 28.05.2021
Handledare: Agneta Eriksson-Granskog

EXAMENSARBETE

Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Jens Henriksson
Arbetets namn:	Funktionell integrationstestning av ett webb-API
Handledare:	Agneta Eriksson-Granskog
Uppdragsgivare:	Crosskey Banking Solutions

Abstrakt

Syftet med det här examensarbetet är att beskriva hur funktionell integrationstestning kan utföras på ett befintligt webb-API.

Resultatet blev en testmiljö som delvis automatiserades med en automationsserver som hanterar ett testprojektet innehållande de funktionella webb-API-testerna.

Nyckelord (sökord)

Automationstestning, funktionell testning, integrationstestning, API-testning, webb-API, C#, xUnit, Jenkins

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2021:34	1458-1531	Svenska	25 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
28.05.2021	12.05.2021	28.05.2021

DEGREE THESIS

Åland University of Applied Sciences

Study program:	Information Technology
Author:	Jens Henriksson
Title:	Functional Integration Testing of a Web API
Academic Supervisor:	Agneta Eriksson-Granskog
Technical Supervisor:	Crosskey Banking Solutions

Abstract

The purpose of this thesis is to describe how functional integration testing can be performed on an existing web API.

The result was a testing environment which was partly automated with an automation server operating a test project containing the functional web API tests.

Keywords

Automation testing, functional testing, integration testing, API testing, Web API, C#, xUnit, Jenkins

Serial number:	ISSN:	Language:	Number of pages:
2021:34	1458-1531	Swedish	25 pages

Handed in:	Date of presentation:	Approved on:
28.05.2021	12.05.2021	28.05.2021

INNEHÅLLSFÖRTECKNING

1. INLEDNING	5
1.1 Syfte	5
1.2 Metod	5
1.3 Avgränsningar	5
1.4 Uppdragsgivare	6
2. BAKGRUND	7
2.1 Varför automationstesta ett webb-API	7
2.2 Webb-API	7
2.2.1 HTTP-förfrågan	7
2.2.2 HTTP-svar	8
2.3 API-testning	8
2.4 Testmönster	9
2.5 Systemet under test	9
2.6 Verktyg	10
2.6.1 Testningsverktyg	10
2.6.2 Automationsserver	10
3. TESTMILJÖN	11
3.1 Skapa en testmiljö	11
3.2 Testdata	12
3.3 Automatisering	12
4. TESTERNA	14
4.1 Testningsstrategi	14
4.2 För alla tester	14
4.3 Test för att hämta data	16
4.3.1 Icke-datadrivet test	16
4.3.2 Datadrivet test	18
4.4 Test för att skapa data	19
4.5 Test för att uppdatera data	20
4.5.1 Testscenario	20
4.5.2 Automatiskt urval av testdata	22
5. SLUTSATSER	23
5.1 Resultat	23
5.2 Framtiden	23
5.3 Reflektioner	23
KÄLLOR	24

1. INLEDNING

1.1 Syfte

Syftet med det här examensarbetet är att ta reda på hur man testar ett befintligt webb-API för att sedan automatisera det och utvärdera resultatet i verkligheten. Arbetet utfördes åt företaget Crosskey.

1.2 Metod

En dedikerad testmiljö behöver finnas för att testerna ska ha något att köra emot. Testmiljöns omfattning och placering måste bestämmas. Testmiljön kommer att delvis automatiseras för att förenkla testningen.

En testningsstrategi kan fastställas då en fungerade testmiljö finns på plats. Webb-API:et eller systemet-under-test saknar till stora delar entydig dokumentation för över hur det fungerar och därmed kommer webb-API:ets gränssnitt (eng. *interfaces*) stå för den huvudsakliga dokumentationen för testningen.

Utveckling av testprojektet kommer att ske, likt systemet-under-test i programmeringsspråket C#. Programutvecklingsmiljön (eng. *integrated development environment*) *Microsoft Visual Studio* kommer tillsammans med testningsramverket *xUnit.net* samt en del bibliotek stå som stomme för testprojektet.

1.3 Avgränsningar

För att begränsa arbetet inom den angivna tidsramen kommer endast funktionella tester att behandlas. Testtäckning (eng. *test coverage*) kommer inte att uppnå 100%. Automatisk generering av tester kommer inte att tas upp. Automatiseringen av testmiljön kommer endast att delvis implementeras då det är i huvudsak testprojektet som står i fokus. Rapportering av testresultatet kommer att skötas av Jenkins och ingen ytterligare utveckling av testrapporter kommer att tas upp.

1.4 Uppdragsgivare

Uppdragsgivaren för det här examensarbetet var Crosskey Banking Solutions. Crosskey är ett it-företag som först och främst levererar olika former av banktjänster. Deras arbetsområde sträcker sig från traditionella banktjänster till bland annat elektroniska banktjänster, kort- och mobilbetalningar, men även kapitalmarknad (*Crosskey*, 2015). Det här arbetet utfördes åt kapitalmarknadsavdelningen.

2. BAKGRUND

2.1 Varför automationstesta ett webb-API

Att testa mjukvara är ofta valfritt. Det kan leda till att man väljer bort att testa, eller ser det som en eftertanke. (*Goals of Test Automation*, u.å.) Ett API är en viktig del av en applikation och står ofta för stora delar av kommunikationen. Funktionella API-tester är vanligtvis snabba att köra och ger en bra avkastning på den investerade tiden som det tar att skapa dem (*REST API Testing Strategy: What Exactly Should You Test?*, 2019).

2.2 Webb-API

Ett webb-API eller webb-applikationsprogrammeringsgränssnitt består av ett antal ändpunkter (eng. *endpoints*). Kommunikation till och från ett webb-API består av förfrågningar (eng. *request*) och svar (eng. *response*). Kommunikationsspråket är vanligtvis *JSON*¹ (Wikipedia contributors, 2021a).

Testningen av webb-API:et sker genom att sammanställa en HTTP-förfrågan som webb-API:et behandlar. HTTP-svaret från webb-API:et är det som testet verifierar att stämmer överens med det förväntade svaret.

2.2.1 HTTP-förfrågan

HTTP eller *Hypertext Transfer Protocol* är en protokoll som tillåter kommunikation mellan klient och server.

En HTTP-förfrågan (eng. *HTTP Request*) kan bestå av dessa HTTP-metoder (*HTTP methods GET vs POST*, u.å.):

- **GET** - Förfrågningar som skickas med GET är till för att hämta data från API:et.
- **POST** - används för att skicka data till API:et och kan användas för att skapa data.
- **PUT** - är till för att uppdatera eller modifiera data genom att ersätta befintlig data.
- **DELETE** - är till för att ta bort befintlig data.

¹ JavaScript Object Notation är en textbaserat språk för att utbyta data (Wikipedia contributors, u.å.).

2.2.2 HTTP-svar

HTTP-svar (eng. *HTTP Response*) är de statuskoder som beskriver en HTTP-förfrågans resultat. En översikt för de statuskoder som behandlas beskrivs i tabell 1.

Tabell 1. De HTTP-statuskoder som används av API:et (*HTTP response status codes, u.å.*)

<i>HTTP-Statuskod</i>	<i>Namn</i>	<i>Förklaring</i>
200	OK	Förfrågan lyckades
201	Created	Den nya resursen har skapats
400	Bad Request	Förfrågan kunde inte hanteras
404	Not Found	Resursen hittades inte

2.3 API-testning

API-testning är en del av mjukvarutestning som fokuserar på att testa API:er. För att testa API:er används ofta integrationstester då de inkluderar hela systemet som ska testas. Ett API kan testas på flera olika sätt (Wikipedia contributors, 2021c):

- Funktionalitet
- Prestanda
- Säkerhet
- Pålitlighet

Integrationstestning är en typ av testning där flera olika komponenter mjukvara testas som ett system. Målet med integrationstestning är att hitta fel då flera komponenter tas i bruk (Wikipedia contributors, 2020a).

Big-bang-testning är en form av integrationstestning där alla komponenter testas som ett system. Det betyder att alla komponenter måste vara på plats och fungera för att testerna ska lyckas. Den här lösningen passar bra för mindre system (Wikipedia contributors, 2020a).

Funktionell testning är en form av *black-box* testning där systemet-under-test kan testas utan att känna till innehållet för hur systemet fungerar (Wikipedia contributors, 2020b).

Med black-box avses ett system där man endast bryr sig om input och output. Hur systemet fungerar i detalj är inte nödvändigt att känna till för att testa (Wikipedia contributors, 2021b).

2.4 Testmönster

Arrange-Act-Assert är ett testmönster som går ut på att dela upp ett test i tre delar.

1. *Arrange* är då man ställer upp allt som behövs för testet. I fallet för API-testning kommer det vara att exempelvis sammanställa en HTTP-förfrågan.
2. *Act* är då man agerar på det man ska testa. I det här fallet kommer det vara att skicka iväg HTTP-förfrågan till web-API:et.
3. *Assert* är då man verifierar att resultatet stämmer överens med det förväntade. I det här fallet kan det vara att HTTP-statuskoden är 200.

En stor fördel med testmönstret är att testerna fokuserar på att testa en sak åt gången och bidrar därmed till ökad läsbarhet samt ett tydligare fokus på testfallet (Knight, 2020).

2.5 Systemet under test

Systemet under test är ett webb-API. I det här fallet är det ett realistiskt att noggrant gå igenom varje ändpunkt och vad de gör. Tabell 2 beskriver webb-API:ets ändpunkter.

Tabell 2. Antal ändpunkter som webb-API:et består av

<i>HTTP-metod</i>	<i>Antal ändpunkter</i>
GET	63
POST	14
PUT	3
DELETE	0
Sammanlagt	80

I tabell 2 visas vilka ändpunkter som webb-API:et består av. Det är i det här fallet väldigt många ändpunkter för att hämta data. Det är ett positivt tecken då de ofta är lättare att förstå och testa än de andra ändpunkterna. I det här fallet fanns det inga ändpunkter för att ta bort någon data.

2.6 Verktyg

2.6.1 Testningsverktyg

xUnit.net eller *xUnit* är ett *open source* testningsverktyg som är byggt inom *.NET Framework*². *xUnit* används vanligtvis för enhetstester (*xUnit.net*, u.å.). Det visar sig däremot vara användbart för funktionella integrationstester.

Det finns två typer av tester i *xUnit*. Det ena benämns med taggen *[Fact]* och är sant oberoende data. Det andra benämns med taggen *[Theory]* och är endast sant för en specifik uppsättning data. För att förse data åt ett teoritest används en datatag, vanligtvis *[InlineData]* (Kanjilal, 2017).

2.6.2 Automationsserver

Jenkins är en *open source* automationsserver som har ett brett användningsområde. Det sträcker sig från byggning, testning, leverans och *deployment* av mjukvara (*Jenkins User Documentation*, u.å.-a).

Jenkins är ett flexibelt verktyg som kan byggas ut med diverse *plug-ins* (*Jenkins User Documentation*, u.å.-b). För att uppnå resultatet jag är ute efter kommer dessa *plug-ins* att användas:

1. Test Results Analyzer
2. Build Monitor View

² Mjukvaruramverk utvecklat av Microsoft primärt för Microsoft Windows (Wikipedia contributors, 2021d).

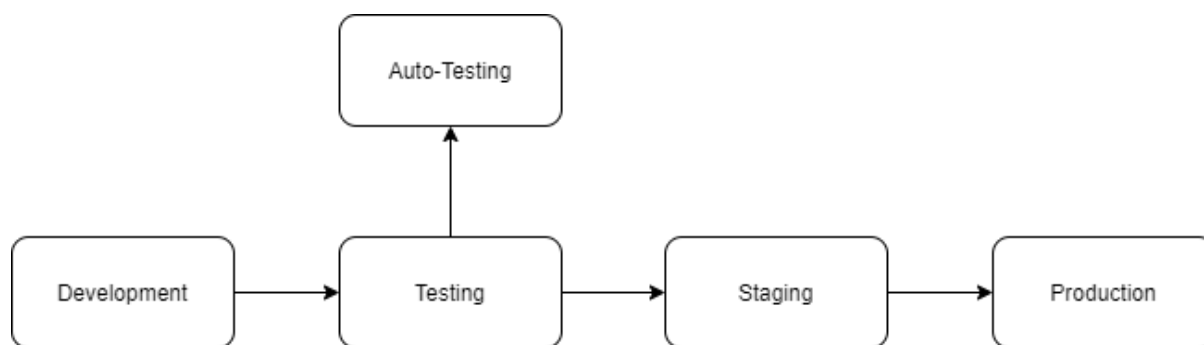
3. TESTMILJÖN

3.1 Skapa en testmiljö

Testmiljön utformades i dialog med kollegor och uppdragsgivare som hade erfarenhet av miljön som skulle testas. Lösningen blev en klon på den befintliga manuella testmiljön vilket i praktiken innebar att en kopia på en befintlig databas skapades. Databasen består av en modifierad *Microsoft Dynamics CRM*³-instans. Webb-API:et fungerar som en datakanal för Dynamics-instansen så att den kan kommunicera med andra tjänster. För att underlätta hanteringen av testmiljön utnyttjades molnlösningen *Azure*⁴.

Testmiljön bör likna en produktionsmiljö för att testerna ska medföra konkreta resultat (Testim, 2019). Det inkluderar externa kopplingar som behövs för att testmiljön ska fungera likt den miljö man försöker replikera. För det här fallet användes en *big-bang-approach* för testmiljön.

Den ursprungliga miljön kan delas upp i fyra olika nivåer vilket illustreras i figur 1.



Figur 1. Testmiljöns förhållande i relation till de andra miljöerna

En utvecklingsmiljö (eng. *development environment*) är avsedd för utveckling och tester på lägre nivå. En *staging*-miljö är däremot avsedd för högre nivåer av tester än de som kommer att användas för det här arbetet (Wikipedia contributors, 2021e).

³ Programvara för kundvård (eng. *Customer relationship management*) utvecklat av Microsoft (Microsoft Corporation, u.å.).

⁴ Molnplattform som erbjuds av Microsoft (*What is Azure—Microsoft Cloud Services*, u.å.).

En stor fördel med att välja en dedikerad testmiljö för automationstester är att undvika störningar från andra som använder miljön samtidigt. Det kan vara människor som använder den manuellt eller andra automatiska tester som kan påverka testernas resultat (Testim, 2019).

3.2 Testdata

Testdatan för testmiljön kom även från den manuella testmiljön. I det här fallet var datan lik den typ av data som kommer att finnas i produktion. Då all data finns i en SQL-databas så är det möjligt att använda den testdatan i testerna.

Testerna kommer att ändra på testdatan, både positivt men även negativt. En del testfall kan införa oönskade kopior eller modifiera data som egentligen inte borde tillåtas. För att medföra konsekventa resultat bör därför testdatan återställas till det ursprungliga tillståndet efter varje testkörning. Det görs genom att manuellt ta en databasbackup på testdatan vid det önskade läget och återställa miljön till det läget vid behov.

3.3 Automatisering

Automatiseringen av testmiljön utfördes med automationsservern Jenkins. I det här fallet är det endast API-testerna som byggs och körs. Automatisering av byggning och installation av testmiljön ingår inte. Testkörningen gjordes med hjälp av ett *dotnet* kommando som körs av Jenkins.

```
dotnet test testprojektet.sln --logger loggfil.xml
```

Där *dotnet test* bygger källkodsfiler för projektet samt exekverar dem. Testresultaten fås ut genom att specificera en testlogger i form av *--logger* som skriver resultaten till en *.xml* fil som kan läsas in i webb-*GUI*:t för Jenkins. Där kan man analysera testresultaten i detalj.

Skeduleringen för när testerna ska köras sköts med hjälp av Jenkins inbyggda skeduleringsfunktion *build periodically*. Med hjälp av det kan man ställa in exakt hur ofta man vill att jobbet ska utföras.

*H 7 * * **

I det här fallet ska testernas byggas och köras varje dag klockan 7 på morgonen.

Återställningen av databasen sker med hjälp av ett *PowerShell*-skript som innehåller kommandot *Add-CrmDatabase*.

```
Add-CrmDatabase -SqlServerName sql -DatabaseName AutoTest_MSCRM  
-DatabaseBackupFileName AutoTest_MSCRM.bak -Credential $Cred -DwsServerUrl  
https://localhost
```

Kommandot *Add-CrmDatabase* används för att återställa en CRM-organisationsdatabas från en databaskopia (eng. *database backup*) innehållande en CRM-organisation med tillhörande data.

4. TESTERNA

4.1 Testningsstrategi

Testningsstrategin som användes gick ut på att skriva funktionella integrationstester för varje ändpunkt. För en del ändpunkter skapades endast ett test som testade att ändpunkten fungerade som förväntat. En del ändpunkter där indata kan variera skapades flera tester med både positiva och negativa utfall. För att tydligt beskriva varje tests ändamål användes en namngivningskonvention som beskrivs i tabell 3.

Tabell 3. Tabell som beskriver namngivning för tester

<i>Testfil</i>	<i>Testmetod</i>	<i>Förklaring</i>
GetAdvisorsTest.cs	GET_Advisors	Test för att hämta data
GetAdvisorsByIdTest.cs	GET_AdvisorById	Test för att hämta specifik data
	GET_AdvisorById_InvalidId	Test för att hämta specifik data med ogiltigt id

I tabell 3 förklaras testprojektets struktur och namngivning. Varje testfil i projektet motsvarar en ändpunkt i webb-API:et. Varje testmetod i testfilen motsvarar ett specifikt fall för en ändpunkt. Varje testmetod kan i praktiken motsvara flera tester eftersom en datatag kan användas som tillåter ett test att köras flera gånger med olika data.

4.2 För alla tester

För alla tester används en och samma basklass för att på ett effektivt sätt konsumera webb-API:et. C#-biblioteket *RestSharp* används för att förenkla denna process. Figur 2 beskriver hur webb-API:et konsumeras av testerna.

```

public abstract class MyAPI
{
    public RestClient RestClient { get; private set; }

    private const string _baseUrl = "http://localhost";

    public MyAPI()
    {
        RestClient = new RestClient(_baseUrl);
    }

    public T ExecuteGet<T>(IRestRequest request) where T : new()
    {
        var response = RestClient.Execute(request);
        if (response.StatusCode == HttpStatusCode.OK)
        {
            return JsonConvert.DeserializeObject<T>(response.Content);
        }

        return default;
    }

    public T Get<T>(string uri) where T : new()
    {
        var request = new RestRequest(uri, Method.GET);
        return ExecuteGet<T>(request);
    }
}

```

Figur 2. Källkod som beskriver hur API:et konsumeras

Basklassen *MyAPI* innehåller en *URL* som innehåller sökvägen till API:et. Det för att alla tester ska köra mot samma adress. *RestSharp* användas i huvudsak för att anropa webbsurser via HTTP-kommunikation. *RestSharp* hjälper även till att serialisera förfrågningar och deserialisera svaren (*Introduction*, 2020).

4.3 Test för att hämta data

4.3.1 Icke-datadrivet test

Ett icke-datadrivet test kan testas utan någon indata. Det är vanligtvis GET-ändpunkter där man ska hämta data. Figur 3 innehåller källkod för ett icke-datadrivet test.

```
namespace ApiTests.Tests.AdvisorService
{
    /// <summary>
    /// GET: /Advisors
    /// </summary>
    /// <returns>A list of advisors.</returns>
    public class GetAdvisorsTest : MyAPI
    {
        [Fact]
        public void GET_Advisors()
        {
            // Arrange
            var request = new RestRequest("advisors", Method.GET);

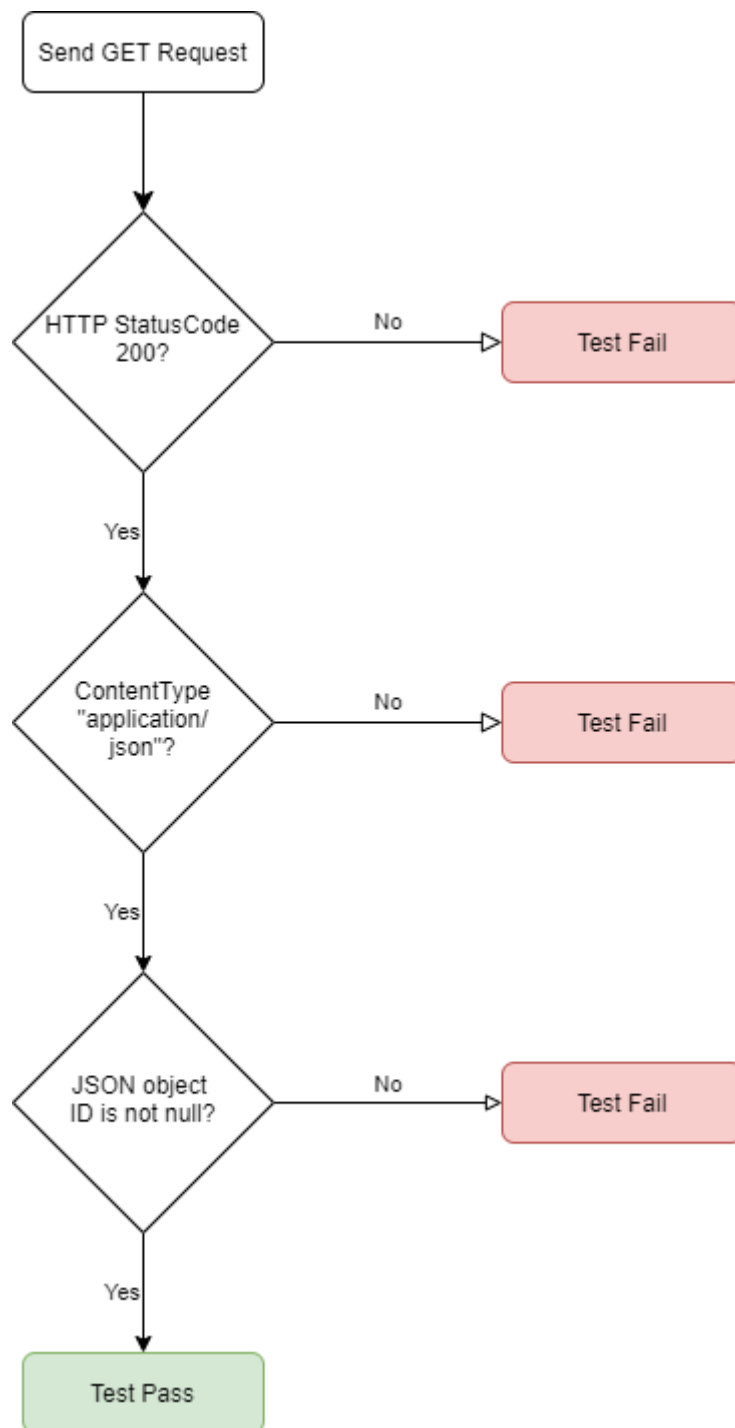
            // Act
            var response = RestClient.Execute(request);
            var advisors = JsonConvert.DeserializeObject<List<Advisor>>(response.Content);

            // Assert
            response.StatusCode.ShouldBe(HttpStatusCode.OK);
            response.ContentType.ShouldContain("application/json");
            advisors.ForEach(advisor => advisor.AdvisorId.ShouldNotBeNullOrEmpty());
        }
    }
}
```

Figur 3. Ett enkelt test för en GET-endpoint

Figur 3 består av källkod som ansvarar för att hantera ett test utan någon specifik testdata och testet är därmed markerat med *[Fact]*, vilket indikerar att testet är sant för alla fall och inte för något specifikt fall.

Processen för att ta fram ett test illustreras i figur 4.



Figur 4. Flödesschema som beskriver processen för att avgöra testresultat för ett GET-anrop

Flödesschemat i figur 4 illustrerar processen för att framställa ett test likt det som visas i figur 3.

4.3.2 Datadrivet test

Datadrivna test kräver någon form av indata. Testet markeras med taggen *[Theory]* samt en data-tagga innehållande indata för testet. I figur 5 visas ett datadrivet test.

```
[Theory]
[InlineData("2054")]
public void GET_AdvisorById(string advisorId)
{
    // Arrange
    var request = new RestRequest($"advisors/{advisorId}", Method.GET);

    // Act
    var response = RestClient.Execute(request);
    var advisors = JsonConvert.DeserializeObject<List<Advisor>>(response.Content);

    // Assert
    response.StatusCode.ShouldBe(HttpStatusCode.OK);
    response.ContentType.ShouldContain("application/json");
    response.Content.Length.ShouldBeGreaterThan(3);
    advisors.FirstOrDefault().AdvisorId.ShouldBe(advisorId);
}
```

Figur 5. Ett positivt datadrivet test för en GET-endpoint

I figur 5 testas en GET-ändpunkt då en specifik *Advisor* efterfrågas via sitt id. Eftersom testet är datadrivet med indata *advisorId* är det förväntade resultatet ett *Advisor*-objekt med samma id som indata, i det här fallet 2054.

Datadrivna test kan även testas med negativa utfall. Ett negativt testfall kan bestå av att använda ett id som inte finns bland testmiljöns testdata eller ett ogiltigt id som inte tillåts. Ett test med negativt utfall visas i figur 6.

```

[Theory]
[InlineData("00000")]
public void GET_AdvisorById_InvalidId(string advisorId)
{
    // Arrange
    var request = new RestRequest($"advisors/{advisorId}", Method.GET);

    // Act
    var response = RestClient.Execute(request);

    // Assert
    response.StatusCode.ShouldBe(HttpStatusCode.NotFound);
    response.ContentType.ShouldContain("application/json");
    response.Content.Length.ShouldBeLessThan(3);
}

```

Figur 6. Ett negativt datadrivet test för en GET-endpoint

I figur 6 används ett ogiltigt id *00000* som inte finns bland testmiljöns testdata och förfrågan bör därmed inte returnera något Advisor-objekt. För att indikera att inget objekt hittades med det efterfrågade id:t används statuskoden *NotFound*.

4.4 Test för att skapa data

Test för att skapa data blir datadrivna och kräver någon form av indata. Det går att testa både positiva och negativa testfall. I figuren nedan illustreras ett positivt testfall. Figur 7 visar ett positivt testfall.

```

[Theory]
[InlineData(1.0, DateTime.Today, "8fc6316e-4702-eb11-a853-000d3aa94200", "Close")]
public void POST_AddData_Price_Valid(double val, DateTime date, string id, string type)
{
    var price = new Price
    {
        Value = val,
        PriceDate = date,
        InstrumentId = id,
        PriceType = type
    };

    var request = new RestRequest("data/Price", Method.POST);
    request.AddJsonBody(new MyRequestBody
    {
        Body = price
    });

    var response = RestClient.Execute(request);
    var result = JsonConvert.DeserializeObject<Price>(response.Content);

    response.StatusCode.ShouldBe(HttpStatusCode.Created);
    response.ContentType.ShouldContain("application/json");
}

```

Figur 7. Testar att skapa data med POST-anrop

I figur 7 testas POST-ändpunkten *data/Price* för att lägga till data för entiteten *Price*. Den ingående testdatan består av prisets värde, datum, identifikation och typ. Entiteten läggs till i en förfrågan och skickas iväg. Då entiteten skapats ska statuskoden *Created* returneras för att indikera att resursen skapades.

4.5 Test för att uppdatera data

För att uppdatera data krävs indata för testet. För det här webb-API:et räcker det med att indatan består av ett id som indikerar vilken data-entitet som ska uppdateras.

4.5.1 Testscenario

Testscenario är test med större omfattning än ett testfall. Ett testscenario motsvarar hur systemet-under-test kan användas i verkligheten då en kedja med händelser uppstår. För ett webb-API kan det innebära att flera ändpunkter används. Figur 8 innehåller ett testscenario.

```

[Theory]
[InlineData("Currency", "b4648cb8-6ebb-ea11-a81c-000d3a2726f0")]
[Trait("Currency", "Update")]
public void PUT_UpdateData_Valid_Currency(string entityType, string guid)
{
    // Scenario: Update AUD exchange rate by 10%
    double amount = 1.1;

    // I. GET current data for Currency AUD
    var initialCurrency = MyAPI.Get<Currency>($"data/{entityType}/{guid}");

    // II. Update the exchange rate
    var newCurrency = Extensions.Clone(initialCurrency);
    newCurrency.ExchangeRate *= amount;

    // III. PUT (Update) the Currency
    var putResponse = MyAPI.Put($"data/{entityType}", new MyRequestBody { Body = newCurrency });

    // IV. GET the updated Currency
    var updatedCurrency = MyAPI.Get<Currency>($"data/{entityType}/{guid}");

    // V. Assert
    updatedCurrency.ExchangeRate.ShouldBe(initialCurrency.ExchangeRate * amount, 0.00001);
    putResponse.StatusCode.ShouldBe(HttpStatusCode.OK);
    putResponse.ContentType.ShouldContain("application/json");
    putResponse.Content.ShouldBe("true");
}

```

Figur 8. Ett testscenario som uppdaterar en valutakurs

Figur 8 beskriver ett testscenario där den australiska dollarns (*AUD*) växlingskurs uppdateras med en 10% ökning gentemot den nuvarande.

Taggen *[Trait]* används för att förtydliga att kategorin för testet är *Currency* vilket motsvarar namnet på entiteten som testas. *Update* indikerar att testet kommer att uppdatera entiteten.

Taggarna hjälper till att organisera testerna då ändpunkten för *UpdateData* kan hantera flera olika entiteter.

Testscenariot har två indataparametrar. Den första är namnet på entiteten och den andra är ett id som motsvarar valutan *AUD* från testdatabasen. Med id:t för valutan hämtas med hjälp av en GET-förfrågan befintlig data för valutan. För att hantera valutan deserialiseras svaret från JSON till ett C#-objekt.

Då startvärdet på valutan har behandlats utförs en *deep copy*⁵ på den hämtade datan. Valutakursen på det klonade objektet uppdateras med en 10 procents ökning. För att uppdatera valutan sammanställs en PUT-förfrågan där *Body* innehåller data för den nya valutan.

⁵ *Deep copy* är en teknik som används för att kлона objekt så att de inte refererar till den ursprungliga datan (Techopedia, 2011).

Det sista steget består av att hämta data för valutan på nytt. Det görs för att verifiera att entiteten faktiskt uppdateras. För att testscenariot ska bli grönt behöver statuskod på PUT-förfrågan, innehållstyp samt innehåll verifieras. I det här fallet returnerar API:et en sträng "true" om förfrågan lyckades.

4.5.2 Automatiskt urval av testdata

En del testfall kan automatiseras genom att automatiskt ta fram testdata från testdatabasen.

Figur 9 visar ett automatiskt urval av testdata.

```
[Theory]
[SqlServerData("SELECT TOP 1 TransactionCurrencyId FROM TransactionCurrency")]
[Trait("Currency", "Update")]
public void PUT_UpdateData_Automatic_Currency(Guid guid)
{
    var request = new RestRequest($"data/Currency", Method.PUT);
    request.AddJsonBody(new MyRequestBody { Body = Get<Currency>($"data/Currency/{guid}") });

    var response = RestClient.Execute(request);

    response.StatusCode.ShouldBe(HttpStatusCode.OK);
    response.ContentType.ShouldContain("application/json");
    response.Content.ShouldContain("true");
}
```

Figur 9. Ett testfall där testdatabasen utnyttjas för att automatisera urvalet av indata

Figur 9 illustrerar ett testfall där databasen används som datakälla för testets indata.

Data-taggen `[SqlServerData(Query)]` frågar testdatabasen efter testdata för entiteten *Currency*.

Frågan (eng. *query*) använder sig av *TOP 1* för att hitta ett id för entiteten *Currency* från tabellen. *TransactionCurrency* motsvarar entiteten *Currency* i testfallet. I det här fallet ändras inte datan under testets gång utan skickas tillbaka i samma tillstånd som det efterfrågades.

5. SLUTSATSER

5.1 Resultat

Syftet med det här examensarbetet är att ta reda på hur man kan testa ett webb-API med funktionalitet i åtanke. Det skedde genom att använda integrationstester då det är i detta sammanhang som webb-API:et kommer att arbeta i produktionsläge.

Testmiljön är en klon på den manuella testmiljön som är helt och hållet dedikerad för automationstester. Testdatan är lik den typ av data som finns i produktionssammanhang. För att automatisera körningen av testerna användes *Jenkins* som bygger, testar, rapporterar, återställer dagligen.

Testerna följer mönstret *Arrange-Act-Assert* som hjälper till att fokusera ett testfall per test. Tydliga namn användes för testerna så att man genom att bara kan läsa testfunktionens namn förstår vilken ändpunkt samt vad som testas. Testerna testar både positiva och negativa utfall. Tydlig skillnad på datadrivna tester och icke-datadrivna tester.

5.2 Framtiden

Testmiljön kan vidareutvecklas genom att lägga till flera tester. Det kan vara andra typer av API-tester som testar annat än funktionalitet. De nya testerna kan exempelvis vara prestandatester, säkerhetstester och tillgänglighetstester. En vidareutveckling på testmiljön i sin helhet kan ske genom *end-to-end* tester som ofta är i form av *GUI*-tester.

5.3 Reflektioner

Det här arbetet har delvis gått ut på att testa ett webb-API som är liknande ett *REST-API* men som egentligen inte följer någon API-spec som exempelvis *OpenAPI*. API:et saknade även till stora delar entydig dokumentation för över hur det fungerade. Det ledde till att genom att använda *PostMan* samt titta på gränssnitten som beskriver ändpunkterna för API:et och delvis frångå *black-box* testning.

KÄLLOR

Crosskey. (2015, januari 2). <https://www.crosskey.fi/>

Goals of Test Automation. (u.å.). Hämtad 03 maj 2021, från

<http://xunitpatterns.com/Goals%20of%20Test%20Automation.html>

HTTP methods GET vs POST. (u.å.). Hämtad 26 april 2021, från

https://www.w3schools.com/tags/ref_httpmethods.asp

HTTP response status codes. (u.å.). Hämtad 26 april 2021, från

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Introduction. (2020). RestSharp. <https://restsharp.dev/getting-started/>

Jenkins User Documentation. (u.å.-a). Hämtad 20 april 2021, från <https://www.jenkins.io/doc/>

Jenkins User Documentation. (u.å.-b). Hämtad 06 maj 2021, från <https://www.jenkins.io/doc/>

Kanjilal, J. (2017, februari 10). *How to work with xUnit.Net framework*.

<https://www.infoworld.com/article/3168787/how-to-work-with-xunit-net-framework.html>

Knight, A. (2020, juli 7). *Arrange-Act-Assert: A pattern for writing good tests*.

<https://automationpanda.com/2020/07/07/arrange-act-assert-a-pattern-for-writing-good-tests/>

Microsoft Corporation. (u.å.). *What is CRM?* Hämtad 27 maj 2021, från

<https://dynamics.microsoft.com/en-us/crm/what-is-crm/>

REST API testing strategy: What exactly should you test? (2019, september 23).

<https://www.sisense.com/blog/rest-api-testing-strategy-what-exactly-should-you-test/>

Techopedia. (2011, december 16). *Deep Copy*. Techopedia.

<https://www.techopedia.com/definition/25608/deep-copy>

Testim. (2019, november 7). *What is a test environment? A guide to managing your testing*.

<https://www.testim.io/blog/test-environment-guide/>

What is Azure—Microsoft Cloud Services. (u.å.). Hämtad 27 maj 2021, från

<https://azure.microsoft.com/en-us/overview/what-is-azure/>

Wikipedia contributors. (u.å.). *JSON*. Wikipedia, The Free Encyclopedia. Hämtad 26 april 2021, från <https://sv.wikipedia.org/w/index.php?title=JSON&oldid=48597361>

Wikipedia contributors. (2020a, december 6). *Integration testing*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Integration_testing&oldid=992642532

Wikipedia contributors. (2020b, december 13). *Functional testing*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Functional_testing&oldid=993974353

Wikipedia contributors. (2021a, januari 5). *Web API*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Web_API&oldid=998501520

Wikipedia contributors. (2021b, januari 12). *Black-box testing*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Black-box_testing&oldid=999915211

Wikipedia contributors. (2021c, februari 10). *API testing*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=API_testing&oldid=1005908121

Wikipedia contributors. (2021d, mars 18). *.NET Framework*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=.NET_Framework&oldid=1012861492

Wikipedia contributors. (2021e, april 7). *Deployment environment*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Deployment_environment&oldid=1016499966

xUnit.net. (u.å.). Hämtad 20 april 2021, från <https://xunit.net/>