Testplan IAC

Dion de Koning

V2C

1. Introductie

Voor deze opdracht moest ik een BMI bereken REST API maken. Deze API moest vervolgens getest worden op verschillende manieren. De service zelf heb ik vrij simpel gehouden. Verder heb ik verschillende geautomatiseerde tests gemaakt.

2. Testomgeving

In dit testdocument ga ik mijn BMICalculator testen. Deze API kan een BMI op basis van je gewicht en lengte in centimeters berekenen. Ik ga de service testen door middel van JUnit en SoapUI. Als je met maven de nieuwste versie van de service build wordt er automatisch een JUnit test uitgevoerd. Deze controleert of de bestaande functionaliteit nog werkt. Wanneer deze slaagt wordt er een root.war gegenereerd. Deze is vervolgens te implementeren op een productieserver. Als de service draait laat ik SoapUI een test suite runnen. Deze stuurt verschillende requests naar de service. Hiervan test de eerste alleen of de service te bereiken is. Daarna wordt de response en de foutafhandeling getest. Deze moeten een verwacht resultaat teruggeven. Gebeurt dit niet dan faalt de test.

3. Smoke test

De smoke test houd in dat als de unit tests slagen de smoke test ook geslaagd is. Dit betekent namelijk dat de service werkt en de reacties correct zijn.

4. Testcases black box test

Voor de black box tests heb ik SoapUI gebruikt. Deze voert de volgende stappen uit:

Stap	Gewicht	Lengte	Uitkomst	Voldaan
Smoketest	64	164	200 Status code	n.v.t.
Normal Test 1	64	164	23.8	n.v.t.
Fault test	0	64	500 Status code	n.v.t.
Normal test 2	60	170	20.8	n.v.t.

Wanneer bovenstaande test slagen is de black box test succesvol. Vervolgens kan er vanuit gegaan worden dat de applicatielogica naar behoren werkt.

5. Unit Tests

De unit tests worden uitgevoerd door JUnit. Dit framework voert een test uit in de buildcycle van maven. Als de tests niet slagen stopt maven met compilen. Omdat het een vrij simpele webservice is is de unittest niet heel uitgebreid. Met de volgende code wordt getest of de functionaliteit van de applicatie nog steeds correct is:

```
public class BMICalcTest {
  private static BMICalc calc;
  @BeforeClass
  public static void initCalc(){
      calc = new BMICalc(164, 64);
  }
  @Before
  public void beforeEachTest(){
      System.out.println("Voor elke test");
  }
  @After
  public void afterEachTest(){
      System.out.println("Na elke test");
  }
  @Test
  public void testCalculate(){
      String result = calc.calculate();
      assertEquals("23.8" , result);
  }
}
```

Daarbij heb ik ook nog een errorTest aangemaakt, daarvan is de code:

```
public class ErrorTest {
    private static Calculate calc;
    @BeforeClass
    public static void initCalc(){
        calc = new Calculate(0, 164);
    }
    @Test
    public void calculate() throws Exception {
        String result = calc.calculate();
```

```
        assertEquals(".0" , result);
    }
}
```

Daarbij is ook de dependency aan maven toegevoegd:

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
```

Elke keer als maven gaat builden wordt de bovenstaande test uitgevoerd. Hiermee wordt vervolgens gekeken of de basis nog steeds werkt.

6. Regressietest

Voor de regressietest heb ik JUnit toegevoegd. Hiermee wordt voordat de applicatie met maven gebuild wordt altijd getest of de vorige functionaliteit nog werkt. Op deze manier zit er altijd nog een check tussen.