****

Testplan

Integration and Communication

CODE TCIF-V2IAC1-15

HBO-ICT SIE Jaar 2 2016-2017

Auteurs

Jochem Kuus

Datum

28 – 01 – 2017

Versie 0.1

© Hogeschool Utrecht, Utrecht, 2015

# Inhoudsopgave

[Inhoudsopgave 2](#_Toc477097601)

[1. Versie 3](#_Toc477097602)

[2. Introductie 3](#_Toc477097603)

[3. Testomgeving 3](#_Toc477097604)

[4. Smoketest 3](#_Toc477097605)

[5. Testcases black box test (integratietest of functionele test) 4](#_Toc477097606)

[6. Unit Tests 4](#_Toc477097607)

# Versie

|  |  |  |
| --- | --- | --- |
| **Versie** | **Wie?** | **Opmerking** |
| 0.1 | Jochem Kuus | Initialisatie |

# Introductie

In dit project wordt er een service gebouwd die aan de hand van een postcode de bijbehorende latitude en longitude opzoekt. De huidige versie wordt beschouwd als versie 1.0, dit omdat de functionaliteit is toegevoegd en de resterende taken alleen maar test gerelateerd zijn.

Er zullen verschillende tests uitgevoerd worden. Er zal doormiddel van JUnit worden ge-unittest, dit wordt gedaan om te controleren dat de Java module correct werkt. JUnit is een simpele manier om dit te testen aangezien zonder al te veel code uitvoerige tests geschreven kunnen worden.

Een andere test die uitgevoerd zal worden is een SOAP test, deze zal worden uitgevoerd in SOAP-UI. In SOAP-UI kan er een ‘testsuite’ aangemaakt worden waarna er testcases gespecificeerd kunnen worden. De testcases worden gerund en daaruit blijkt (als de unit-test al is geweest) dat naast de Java-module ook de communicatie met de module en de Web-Service in orde is.

# Testomgeving

De enige service die getest gaat worden is de LocationService. Allereerst is er de vraag of de service correct werkt, om vervolgens te controleren of de opgelegd beperkingen niet overschreden kunnen worden. Er is een lengte beperking (maximale lengte is 6), een samenstellingsbeperking (4 cijfers en 2 letters in die volgorde) en er is een beperking in het resultaat (er moet wel een locatie terugkomen). Al deze beperkingen horen foutcodes terug te geven die dan bij SOAP-UI worden getoond.

# Smoketest

De volgende testcases worden uitgevoerd:

1. (Correct) Postcode = 3731XC
2. (Correct) Postcode = 3731xc
3. (Incorrect makeup/lengte) Postcode = 33731XC
4. (Incorrect makeup/lengte) Postcode = 731XC (In JUnit als 3731X)
5. (Incorrect bestaat niet) Postcode = 6842RR (In JUnit als 5555RR)
6. (Incorrect makeup) Postcode = 37317X
7. (Incorrect makeup) Postcode = @373XC (In JUnit als 3731@X)

In mijn beleving zijn deze tests voldoende om alle functionaliteit van het systeem te controleren.

# Testcases black box test

|  |  |
| --- | --- |
| Request | Response |
| <req:locationRequest>3731XC</req:locationRequest> | Een lat- en longitude. |
| <req:locationRequest>3731xc</req:locationRequest> | Een lat- en longitude. |
| <req:locationRequest>33731XC</req:locationRequest> | Verkeerde markup/lengte. |
| <req:locationRequest>731XC</req:locationRequest> | Verkeerde markup/lengte. |
| <req:locationRequest>6842RR</req:locationRequest> | Locatie bestaat niet. |
| <req:locationRequest>37317X</req:locationRequest> | Verkeerde markup/lengte. |
| < req:locationRequest>@373XC</req:locationRequest> | Verkeerde markup/lengte. |

# Unit Tests

JUnit testcases:

|  |  |
| --- | --- |
| Sourcecode | Testcase |
| public class posTest {  // test in which the result will be correct, postcode found and lat/long are matching  @Test  public void classWorks() throws Inputfault {  LocationServiceImpl lSI = new LocationServiceImpl();  Response result = lSI.locationService("3731XC");  BigDecimal resultLat = result.getLatitude();  BigDecimal resultLng = result.getLongitude();  assertEquals(resultLat.doubleValue(), 52.1113981, 4);  assertEquals(resultLng.doubleValue(), 5.1796013, 4);  }  } | 1 |
| public class negTestInvalLeng {  // testing with a postcode of an invalid length  @Test(expected = Inputfault.class)  public void classCrashLen() throws Inputfault {  LocationServiceImpl lSI = new LocationServiceImpl();  lSI.locationService("33731XC");  }  } | 3 |
| public class negTestManyNums {  // testing with a postcode that is incorrect  @Test(expected = Inputfault.class)  public void classCrashMoreNums() throws Inputfault {  LocationServiceImpl lSI = new LocationServiceImpl();  lSI.locationService("37317X"); // this postcode has 5 numbers and 1 letter instead of 4 numbers and 2 letters  }  } | 6 |
| public class negTestInvalChar {  // testing with a postcode that is incorrect  @Test(expected = Inputfault.class)  public void classCrashInvalidChar() throws Inputfault {  LocationServiceImpl lSI = new LocationServiceImpl();  lSI.locationService("3731@X"); // this postcode has a @ which is an illegal character  }  } | 7 |
| public class negTestNotExist {  // testing with a postcode that doesn't exist  @Test(expected = Inputfault.class)  public void classCrashNoLoc() throws Inputfault {  LocationServiceImpl lSI = new LocationServiceImpl();  lSI.locationService("5555RR");  }  } | 5 |