[FahrradKonfigurator](https://github.com/mcmyffin/SE2P_FahrradKonfigurator/tree/c223092ece04b9c761b5676c42f23af4833b1205)

im Auftrag der Firma

Stefan Sarstedt

**Aufgabenblatt 3**

Henning K., Torben-Dennis M. , Saeed S. , Dimitri M. , Abdessamad A.

Stand: 26.05.15

**Inhaltsverzeichnis**

[1. Aufgabe1:Projektbericht](#__RefHeading___Toc448_1398524040)

[1.1. Status des Projekts](#__RefHeading___Toc450_1398524040)

[1.2. Fachliches Datenmodell](#__RefHeading___Toc450_1398524040)

[1.3.Architektur des Gesamtsystems](#__RefHeading___Toc450_1398524040)

[2. Aufgabe2:Risikomanagement](#__RefHeading___Toc662_695674074)

[2.1. Resikoliste](#__RefHeading___Toc664_695674074)

[2.2. Verantwortlicher für das Resikomanagement](#__RefHeading___Toc666_695674074)

[2.3. Frühres Risikomanagement](#__RefHeading___Toc1112_695674074)

[3. Aufgabe3:QS/Testen](#__RefHeading___Toc668_695674074)

3-1BDKundeTest

3-2-TeilKomponente

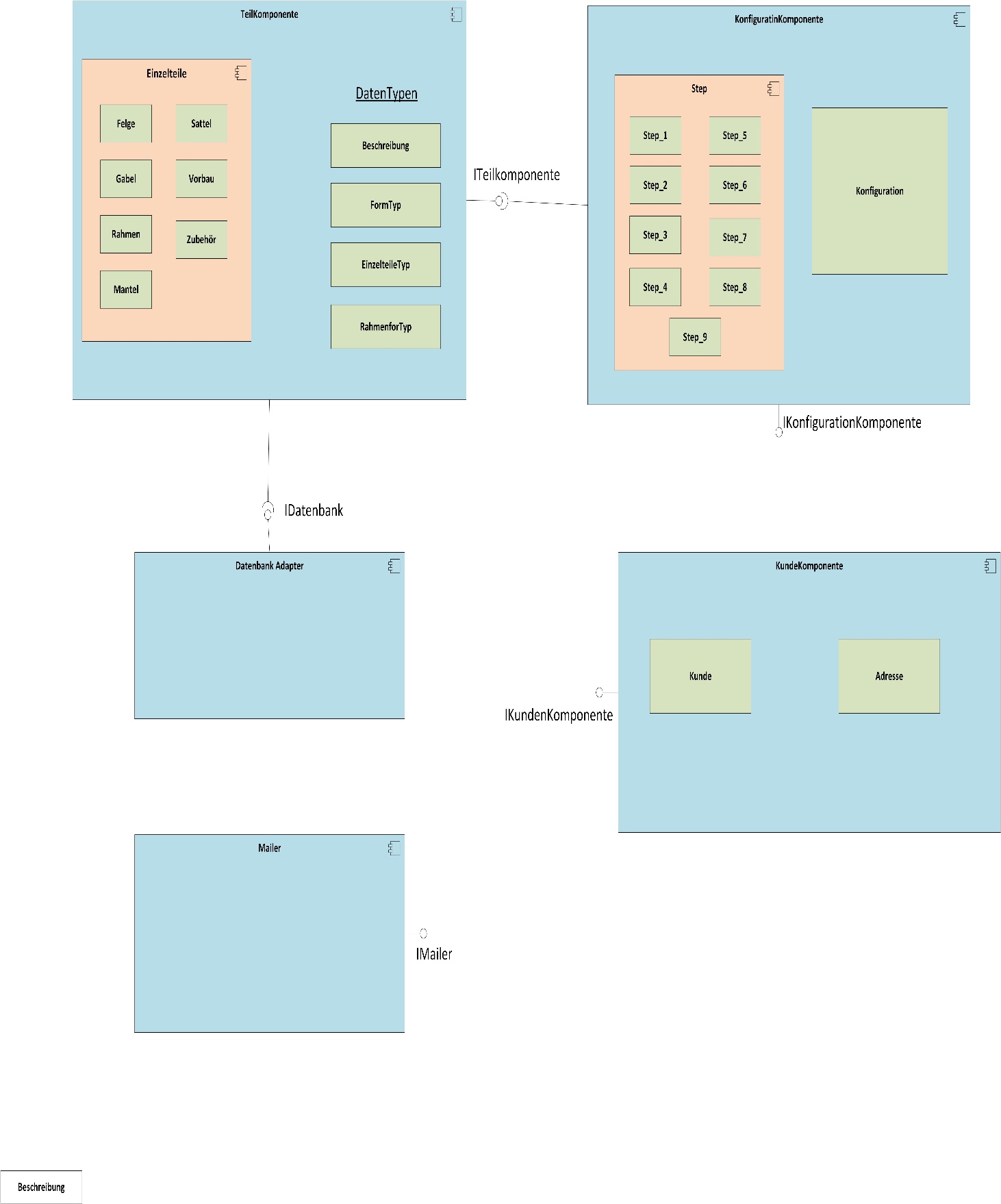
3-2-1 EinzelteileTypTest

3-2-2 ValuesTest

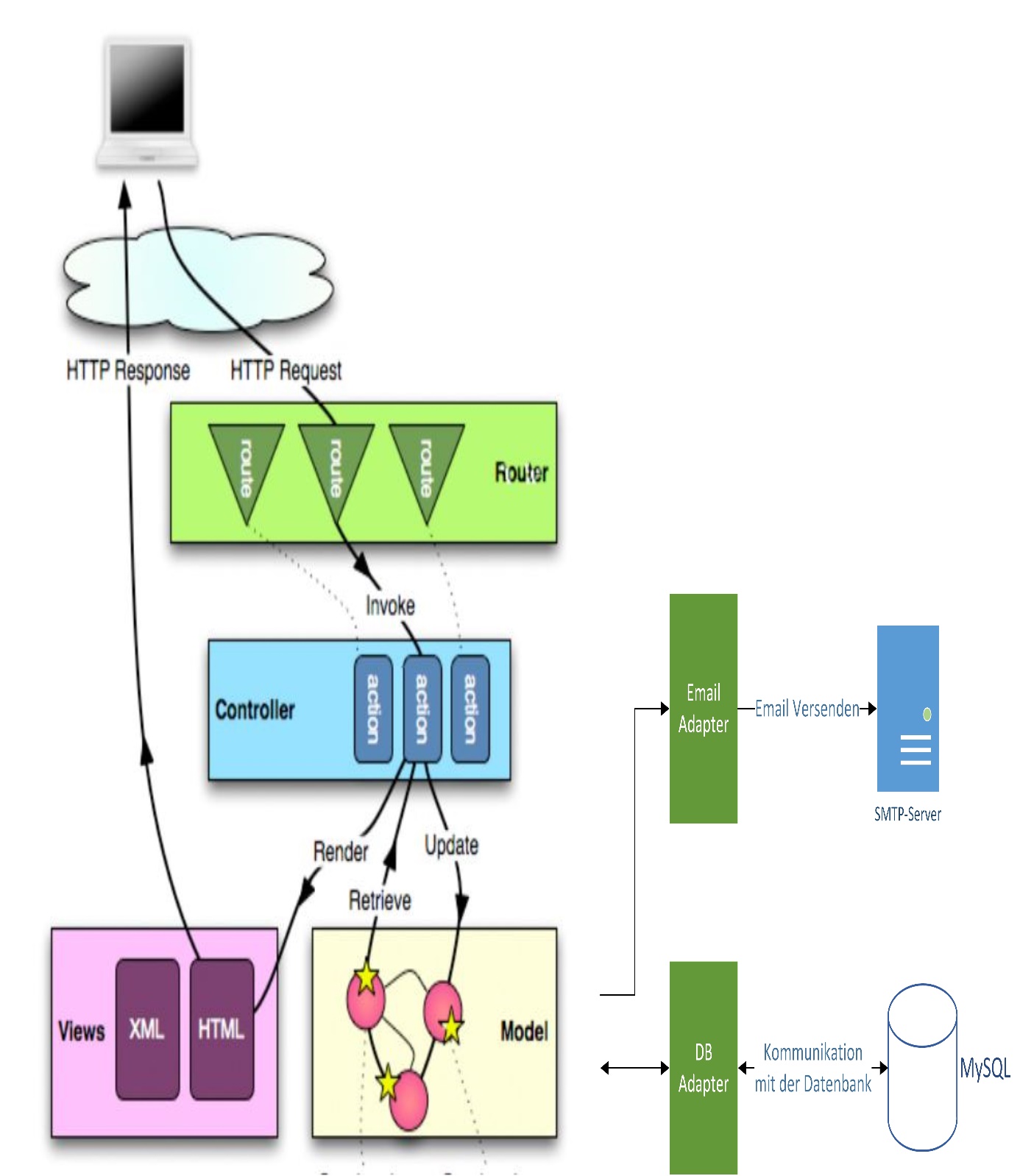
**Status des Projekts**

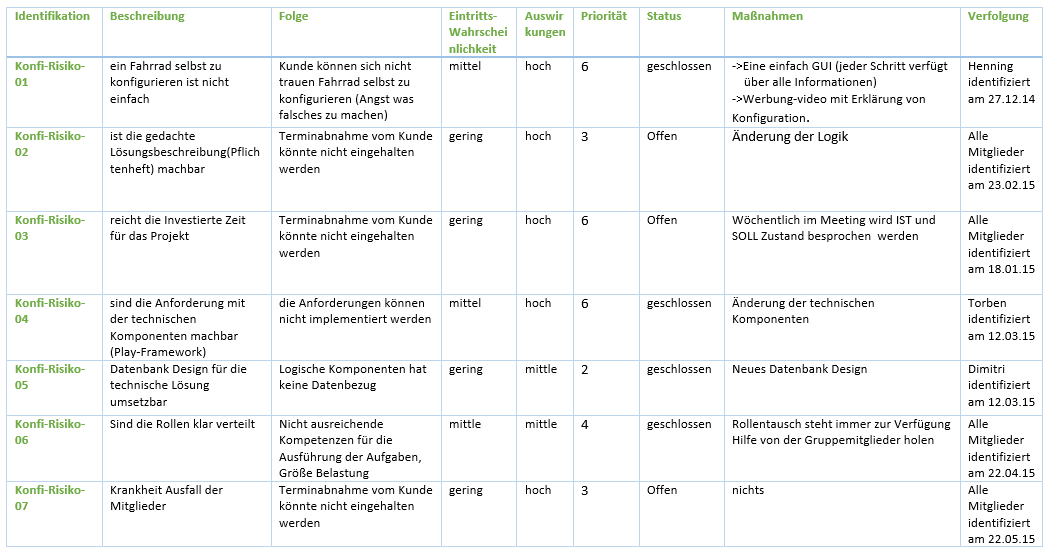
* Front-End:
  + Zum Größtenteils Fertig
* Back-End:
  + Zum Größtenteils Fertig
* Kommunikation zwischen Backend und Frontend:
  + In Bearbeitung
* Nähere Beschreibung:
  + Siehe Kanban

**Fachliches Datenmodell**

****

**Systemarchitektur**

****

**Risikomanagement**

**Verantwortlicher für das Resikomanagement**

Saeed Shanidar

**Früheres Risikomanagement**

Durch die Einführung von dem Risikomanagement hätten wir nur paar Probleme früher erkannt ZB technische Komponente, weil die anderen Risiken wurden von Teilnehmer schon in Anspruch genommen sowie Zeit-Risiko.

**Tests**

**DBKundeTest**

import junit.framework.TestCase;

import models.DatenBank.DataC;

import models.DatenBank.IDatenBank;

import org.junit.Before;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import com.fasterxml.jackson.databind.JsonNode;

import org.junit.\*;

import play.mvc.\*;

import play.test.\*;

import play.data.DynamicForm;

import play.data.validation.ValidationError;

import play.data.validation.Constraints.RequiredValidator;

import play.i18n.Lang;

import play.libs.F;

import play.libs.F.\*;

import play.twirl.api.Content;

import static play.test.Helpers.\*;

import static org.fest.assertions.Assertions.\*;

import java.sql.Connection;

import java.util.ArrayList;

import java.util.List;

import play.db.\*;

/\*\*

\* Created by tin on 30.04.15.

\*/

public class DBKundeTest extends TestCase {

IDatenBank \_db1 = new DataC();

@Before

public void setUp(){

}

public void testIsKundeExistByMail() throws Exception {

//Assert True evaluation

assertTrue(\_db1.isKundeExistByMail("dada@dumdum.de"));

//Assert False evaluation

assertFalse(\_db1.isKundeExistByMail("hullu"));

assertFalse(\_db1.isKundeExistByMail("12345"));

assertFalse(\_db1.isKundeExistByMail(""));

assertFalse(\_db1.isKundeExistByMail("hihihih"));

}

public void testGetKundeByID() throws Exception {

//Eine testliste erstellen mit den gegebenen werten, die in der Datenbank stehen

List<String> testList= new ArrayList<String>();

List<String> testList2= new ArrayList<String>();

testList.add("1");

testList.add("dada@dumdum.de");

testList.add("Dumdum");

testList.add("123");

testList.add("Dada");

//Assert True evaluation

assertEquals(testList,\_db1.getKundeByID(1));

assertEquals(testList2, \_db1.getKundeByID(0));

//Assert False evaluation

assertNotSame(testList,\_db1.getKundeByID(0));

assertNotSame(testList2,\_db1.getKundeByID(1));

}

public void testGetKundeIDByLogin() throws Exception {

//Assert True evaluation

assertEquals(1, \_db1.getKundeIDByLogin("dada@dumdum.de", "kenntKeiner"));

assertEquals(0, \_db1.getKundeIDByLogin("Halluuuuuu", "heehehhehe"));

//Assert False evaluation

assertNotSame(0,\_db1.getKundeIDByLogin("dada@dumdum.de","kenntKeiner"));

assertNotSame(12, \_db1.getKundeIDByLogin("dada@dumdum.de", "kenntKeiner"));

assertNotSame(1,\_db1.getKundeIDByLogin("", ""));

}

public void testGetAdressByID() throws Exception {

//Erstellen einiger Testlisten

List<String> testList = new ArrayList<String>();

List<String> testList2 = new ArrayList<String>();

testList.add("Dada Straße");

testList.add("1");

testList.add("10000");

testList.add("Dadamstadt");

testList.add("Damnit!");

//Assert True evaluation

assertEquals(testList,\_db1.getAdressByID(1));

assertEquals(testList2, \_db1.getAdressByID(0));

//Assert False evaluation

assertNotSame(testList, \_db1.getAdressByID(0));

assertNotSame(testList2, \_db1.getAdressByID(1));

}

public void testSetVorname() throws Exception {

//Assert True evaluation

assertTrue(\_db1.setVorname(1, "Dada"));

//Assert False evaluation

assertFalse(\_db1.setVorname(0, "Dada"));

assertFalse(\_db1.setVorname(1, ""));

}

public void testSetNachname() throws Exception {

//Assert True evaluation

assertTrue(\_db1.setNachname(1, "Dumdum"));

//Assert False evaluation

assertFalse(\_db1.setNachname(0, "Dada"));

assertFalse(\_db1.setNachname(1, ""));

}

public void testSetAdresse() throws Exception {

//Erzeugen einiger Test listen

List<String> testList = new ArrayList<String>();

List<String> testList2 = new ArrayList<String>();

testList.add("Dada Straße");

testList.add("1");

testList.add("10000");

testList.add("Dadamstadt");

testList.add("Damnit!");

//Assert True evaluation

assertTrue(\_db1.setAdresse(1, testList));

//Assert False evaluation

assertFalse(\_db1.setAdresse(0, testList));

assertFalse(\_db1.setAdresse(1, testList2));

}

public void testSetEMail() throws Exception {

//Assert True evaluation

assertTrue(\_db1.setEMail(1, "dada@dumdum.de"));

//Assert False evaluation

assertFalse(\_db1.setEMail(1,""));

assertFalse(\_db1.setEMail(0,"heeeehehhe"));

}

public void testSetPasswort() throws Exception {

//Assert True evaluation

assertTrue(\_db1.setPasswort(1, "kenntKeiner"));

//Assert False evaluation

assertFalse(\_db1.setPasswort(1, ""));

assertFalse(\_db1.setPasswort(0,"aasdgfar"));

}

}

**TeileKomponente**

**EinzelteileTypTest**

package TeileKomponente;

import junit.framework.TestCase;

import models.Exception.EinzelteileTypException;

import models.TeileKomponente.EinzelTeileTyp;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

/\*\*

\* Created by dima on 25.05.15.

\*/

public class EinzelteileTypTest {

@Test

public void testGetEinzelteileTypByStringPositiv() throws EinzelteileTypException {

// Positive Tests

assertEquals(EinzelTeileTyp.FELGE, EinzelTeileTyp.getEinzelteileTypByString("Felge"));

assertEquals(EinzelTeileTyp.GABEL, EinzelTeileTyp.getEinzelteileTypByString("Gabel"));

assertEquals(EinzelTeileTyp.MANTEL, EinzelTeileTyp.getEinzelteileTypByString("Mantel"));

assertEquals(EinzelTeileTyp.RAHMEN, EinzelTeileTyp.getEinzelteileTypByString("Rahmen"));

assertEquals(EinzelTeileTyp.SATTEL, EinzelTeileTyp.getEinzelteileTypByString("Sattel"));

assertEquals(EinzelTeileTyp.VORBAU, EinzelTeileTyp.getEinzelteileTypByString("Vorbau"));

}

@Test(expected = EinzelteileTypException.class)

public void testGetEinzelteileTypByStringGrenzwerte() throws EinzelteileTypException {

//negative Tests

EinzelTeileTyp.getEinzelteileTypByString(null);

EinzelTeileTyp.getEinzelteileTypByString("");

EinzelTeileTyp.getEinzelteileTypByString("felge");

EinzelTeileTyp.getEinzelteileTypByString("gabel");

EinzelTeileTyp.getEinzelteileTypByString("mantel");

EinzelTeileTyp.getEinzelteileTypByString("rahmen");

EinzelTeileTyp.getEinzelteileTypByString("sattel");

EinzelTeileTyp.getEinzelteileTypByString("vorbau");

}

}

**ValuesTest**

package TeileKomponente;

import models.Exception.EinzelteileTypException;

import models.Exception.ValuesCreateException;

import models.TeileKomponente.EinzelTeileTyp;

import models.TeileKomponente.Einzelteile.Values;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

/\*\*

\* Created by dima on 25.05.15.

\*/

public class ValuesTest {

@BeforeClass

public void beforeClass(){

}

// Positiver Test auf Uebereinstimmung der EinzelteileTypen

@Test

public void testValuesAufEinzelteileTyp() throws ValuesCreateException {

int id = 1;

int formTypID = 1;

int beschreibungID = 1;

int hoehe = 10;

int form = 1;

int reifengroesse = 100;

int steuersatz = 22;

int schaftshoehe = 13;

int zusatzbefest = 12;

int tretlager = 18;

int formTyp = 1;

int preis = 100;

int schaftslaenge = 1;

int durchmesser = 2;

String name = "mokupName";

String bild = "/mokupPfad/mokupBild.png";

boolean scheibenbremse = true;

boolean felgenbremse = true;

boolean gepaektraeger = true;

boolean licht = true;

boolean narbenDynamo = true;

boolean zusatzBeleg = true;

// Positive Tests

assertEquals(Values.createFelge(id,beschreibungID,preis,reifengroesse,formTyp,formTypID,bild,name,narbenDynamo).getEinzelTeileTyp() , EinzelTeileTyp.FELGE);

assertEquals(Values.createGabel(id, beschreibungID, formTyp, formTypID, schaftslaenge, steuersatz, reifengroesse, preis,

felgenbremse, scheibenbremse, licht, bild, name).getEinzelTeileTyp() , EinzelTeileTyp.GABEL);

assertEquals(Values.createMantel(id, beschreibungID, formTyp, formTypID, durchmesser, reifengroesse, preis, bild,

name).getEinzelTeileTyp(), EinzelTeileTyp.MANTEL);

assertEquals(Values.createRahmen(id, hoehe, form, reifengroesse, bild, beschreibungID, steuersatz, schaftshoehe, zusatzbefest, scheibenbremse,

felgenbremse, tretlager, formTyp, formTypID, name, preis, gepaektraeger, licht).getEinzelTeileTyp(), EinzelTeileTyp.RAHMEN);

assertEquals(Values.createSattel(id, name, beschreibungID, preis, formTyp, formTypID, bild).getEinzelTeileTyp(), EinzelTeileTyp.SATTEL);

assertEquals(Values.createVorbau(id, beschreibungID, preis, schaftshoehe, formTyp, formTypID, bild).getEinzelTeileTyp(), EinzelTeileTyp.VORBAU);

assertEquals(Values.createZubehoer(id,beschreibungID,preis,formTyp,formTypID,bild,name,zusatzBeleg).getEinzelTeileTyp(), EinzelTeileTyp.ZUBEHOER);

}

// Negativer Test auf ungueltige ID's

@Test(expected = ValuesCreateException.class)

public void testValuesAufUngueltigeIDs\_0() throws ValuesCreateException {

int id = 0;

int formTypID = 0;

int beschreibungID = 0;

int hoehe = 10;

int form = 1;

int reifengroesse = 100;

int steuersatz = 22;

int schaftshoehe = 13;

int zusatzbefest = 12;

int tretlager = 18;

int formTyp = 1;

int preis = 100;

int schaftslaenge = 1;

int durchmesser = 2;

String name = "mokupName";

String bild = "/mokupPfad/mokupBild.png";

boolean scheibenbremse = true;

boolean felgenbremse = true;

boolean gepaektraeger = true;

boolean licht = true;

boolean narbenDynamo = true;

boolean zusatzBeleg = true;

// Positive Tests

assertEquals(Values.createFelge(id,beschreibungID,preis,reifengroesse,formTyp,formTypID,bild,name,narbenDynamo).getEinzelTeileTyp() , EinzelTeileTyp.FELGE);

assertEquals(Values.createGabel(id, beschreibungID, formTyp, formTypID, schaftslaenge, steuersatz, reifengroesse, preis,

felgenbremse, scheibenbremse, licht, bild, name).getEinzelTeileTyp() , EinzelTeileTyp.GABEL);

assertEquals(Values.createMantel(id, beschreibungID, formTyp, formTypID, durchmesser, reifengroesse, preis, bild,

name).getEinzelTeileTyp(), EinzelTeileTyp.MANTEL);

assertEquals(Values.createRahmen(id, hoehe, form, reifengroesse, bild, beschreibungID, steuersatz, schaftshoehe, zusatzbefest, scheibenbremse,

felgenbremse, tretlager, formTyp, formTypID, name, preis, gepaektraeger, licht).getEinzelTeileTyp(), EinzelTeileTyp.RAHMEN);

assertEquals(Values.createSattel(id, name, beschreibungID, preis, formTyp, formTypID, bild).getEinzelTeileTyp(), EinzelTeileTyp.SATTEL);

assertEquals(Values.createVorbau(id, beschreibungID, preis, schaftshoehe, formTyp, formTypID, bild).getEinzelTeileTyp(), EinzelTeileTyp.VORBAU);

assertEquals(Values.createZubehoer(id,beschreibungID,preis,formTyp,formTypID,bild,name,zusatzBeleg).getEinzelTeileTyp(), EinzelTeileTyp.ZUBEHOER);

}

// Negativer Test auf ungueltige ID's

@Test(expected = ValuesCreateException.class)

public void testValuesAufUngueltigeIDs\_1() throws ValuesCreateException {

int id = -1;

int formTypID = -1;

int beschreibungID = -1;

int hoehe = 10;

int form = 1;

int reifengroesse = 100;

int steuersatz = 22;

int schaftshoehe = 13;

int zusatzbefest = 12;

int tretlager = 18;

int formTyp = 1;

int preis = 100;

int schaftslaenge = 1;

int durchmesser = 2;

String name = "mokupName";

String bild = "/mokupPfad/mokupBild.png";

boolean scheibenbremse = true;

boolean felgenbremse = true;

boolean gepaektraeger = true;

boolean licht = true;

boolean narbenDynamo = true;

boolean zusatzBeleg = true;

// Positive Tests

assertEquals(Values.createFelge(id,beschreibungID,preis,reifengroesse,formTyp,formTypID,bild,name,narbenDynamo).getEinzelTeileTyp() , EinzelTeileTyp.FELGE);

assertEquals(Values.createGabel(id, beschreibungID, formTyp, formTypID, schaftslaenge, steuersatz, reifengroesse, preis,

felgenbremse, scheibenbremse, licht, bild, name).getEinzelTeileTyp() , EinzelTeileTyp.GABEL);

assertEquals(Values.createMantel(id, beschreibungID, formTyp, formTypID, durchmesser, reifengroesse, preis, bild,

name).getEinzelTeileTyp(), EinzelTeileTyp.MANTEL);

assertEquals(Values.createRahmen(id, hoehe, form, reifengroesse, bild, beschreibungID, steuersatz, schaftshoehe, zusatzbefest, scheibenbremse,

felgenbremse, tretlager, formTyp, formTypID, name, preis, gepaektraeger, licht).getEinzelTeileTyp(), EinzelTeileTyp.RAHMEN);

assertEquals(Values.createSattel(id, name, beschreibungID, preis, formTyp, formTypID, bild).getEinzelTeileTyp(), EinzelTeileTyp.SATTEL);

assertEquals(Values.createVorbau(id, beschreibungID, preis, schaftshoehe, formTyp, formTypID, bild).getEinzelTeileTyp(), EinzelTeileTyp.VORBAU);

assertEquals(Values.createZubehoer(id,beschreibungID,preis,formTyp,formTypID,bild,name,zusatzBeleg).getEinzelTeileTyp(), EinzelTeileTyp.ZUBEHOER);

}

}