DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Duale Hochschule Baden-Württemberg
Mannheim

Seminararbeit

Fallstudie

Studiengang Wirtschaftsinformatik

Studienrichtung Software Engineering

Verfasserin:	Erika Musterfrau
Matrikelnummer:	123456
Firma:	Musterfirma 123
Abteilung:	Softwareentwicklung
Kurs:	WWI 99 SEZ
Studiengangsleiter:	Prof. Dr. Julian Reichwald
Wissenschaftlicher Betreuer:	Dr. Max Mustermann test@test.com +49 151 / 123 456
Firmenbetreuer:	Moritz Testname test@test.com +49 151 / 123 456
Bearbeitungszeitraum:	01.01.1970 – 31.12.2099

Arbeitsaufteilung

Markus Böbel
- Aufsetzen und Initialisierung des Tomcat-Servers
- Entwurf und Implementierung der RESTful Schnittstelle
- Umsetzung der Authentifizierung
- Implementierung des Grundgerüstes der Game-Klasse
- Implementierung des Soziale- Leistungen- Feature
- Einbau der Frontendlogik mit Hilfe von Angular2

Markus Böbel
- Aufsetzen und Initialisierung des Tomcat-Servers
- Entwurf und Implementierung der RESTful Schnittstelle
- Umsetzung der Authentifizierung
- Implementierung des Grundgerüstes der Game-Klasse
- Implementierung des Soziale- Leistungen- Feature
- Einbau der Frontendlogik mit Hilfe von Angular2

Inhaltsverzeichnis

Abbildungsverzeichnis **iv**

Tabellenverzeichnis **v**

Quelltextverzeichnis **vi**

Abkürzungsverzeichnis **vii**

1 Einleitung Niklas Schuster **1**

1.1 Aufgabenstellung 1

1.2 Zielsetzung 1

2 Entwurf **2**

2.1 Konzept Niklas Schuster 2

2.1.1 Spieldynamik 2

2.1.2 Echtzeit & Ranking 3

2.2 Architektur Markus Böbel 4

3 Entwicklung **6**

3.1 Java Backend 6

3.1.1 Die Klasse „Game“ 6

3.1.2 Das Package „Unternehmung“ 8

3.2 Rest-Schnittstelle Markus Böbel 20

3.2.1 Token - Autorisierung 22

3.3 Angular JS Markus Böbel 23

3.3.1 Aufbau einer Angular 2 Anwendung 24

3.3.2 Komponente der Produktion 26

4 Tests **31**

4.1 Ziel des Testens 31

4.2 Testen mit JUnit 31

4.3 User Akzeptanztest 32
4.3.1 Erstes Szenario 32
4.3.2 Zweites Szenario 33
4.3.3 Drittes Szenario 33
4.3.4 Nutzerbeschreibung und Feedback 34

A Testanhang **35**
A.1 Subtestanhang 36

B Noch ein Testanhang **37**

Abbildungsverzeichnis

Abbildung 2.1 Aufbau der Applikationsarchitektur 5

Abbildung 3.1 Aufbau einer Angular 2 Applikation 25
Abbildung 3.2 Einteilung der Produktionskomponente 27

Tabellenverzeichnis

Quelltextverzeichnis

3.1 `authentivateUser() Methode der GameController-Klasse` 23
3.2 Beispiel des `@Secured`-Dekorators 23
3.3 Grundaufbau einer Komponente 26
3.4 Ausschnitt des Templates der Produktionskomponente 28
3.5 Beispiel für das Anzeigen von Daten in Angular 29
3.6 Anfragen von Daten 29

Abkürzungsverzeichnis

REST Representational State Transfer

HTTP Hyper Text Transfer

SPA Single Page Application

MVC Model-View-Controller

1 Einleitung Niklas Schuster

1.1 Aufgabenstellung

Im Rahmen dieses Projektes, welches im Zuge der Lehrveranstaltung Fallstudie des Moduls Umsetzung der Methoden der Wirtschaftsinformatik erfolgt, soll ein computergestütztes Unternehmensplanspiel entwickelt werden. Dabei soll es sich um ein zu führendes Industrieunternehmen handeln und branchenspezifische Unternehmensprozesse modellhaft simulieren. Weiterhin vorgesehen ist, dass teilnehmende Spieler jeweils ein einzelnes Unternehmen führen sollen und untereinander auf einem Oligopolmarkt konkurrieren. Dafür ist jedes Unternehmen der selben Branche zugeordnet und produziert Produkte in direkter Konkurrenz.

1.2 Zielsetzung

Das Ziel dieser Ausarbeitung ist es, ein funktionierendes Planspiel zu entwickeln, welches in der Programmiersprache Java implementiert wird. Das Planspiel wird den Namen EARTHBOUND tragen und die Führung eines Unternehmens in der Outdoor Branche widerspiegeln, welches auf Rucksäcke, Taschen und ähnliches spezialisiert ist. Als GUI dient dem Spieler ein Web Interface. Dieses wird mit Angular.js und Bootstrap geschrieben. Ziel ist es dem Spieler Unternehmensprozesse in den Bereichen: Human Resources, Produktion, Sales, Marketing und Research zu vermittelt. Besonders großen Augenmerk liegt auf dem planerischen Aspekt des zu führenden Unternehmens sowie das Zeit- und Ressourcen Managements.

2 Entwurf

2.1 Konzept Niklas Schuster

2.1.1 Spieldynamik

Die Spieldynamik des zu entwickelnden Planspiels wird gesteuert und bestimmt durch ein Double-Layer Ansatz welcher das Kernelement des Spiels darstellt und folgende logische Schichten beinhaltet:

1. Die erste Schicht beinhaltet sämtliche funktionale Spielinhalte wie die einzelnen Abteilungen und Features, die für die Spieler spielbar sind.

2. Die zweite Schicht umfasst ein umfangreichen Zahlenpool, welcher aus monetären und nicht monetären Kennzahlen besteht und bestimmt ist die Entscheidungen des Spielers effektiver und realistischer auf das Spielgeschehen abzubilden.

Entscheidend hierbei ist, dass jede Aktion im Spiel die der Spieler ausführt direkten Einfluss auf den Erfolg des zu führenden Unternehmens hat. Dies wird an folgendem Beispiel deutlich:

In der Abteilung Human Ressource hat der Spieler die Möglichkeit Mitarbeiter für sein Unternehmen einzustellen. Ein Mitarbeiter ist im Spiel ein Objekt welches verschiedene Attribute aufweist. So bekommt ein Mitarbeiter z.B. ein Gehalt, sowie eine Abteilung die ihm vom Spieler zugewiesen beziehungsweise als Eingabe durch den Spieler frei wählbar ist. Das Gehalt was den Mitarbeitern bezahlt wird beeinflusst der Höhe nach die Kennzahl Mitarbeiterzufriedenheit und andere Kennzahlen aus dem Zahlenpool beeinflusst. Der Durchschnitt aller Kennzahlen ist ein Indikator für die Marge die sich beim Verkauf von Produkten entsteht. Ein Unternehmen mit zufriedenen Mitarbeitern und einem guten Image kann also Produkte zum selben Herstellungspreis teurer Verkaufen. Somit ergeben sich automatisch aus dem Führungsstil des Spielers unterschiedliche Unternehmensstrategien wie die Kostenführerschaft oder Differenzierung. Wenn der Spieler nur einen geringen Durchschnitt aller

Kennzahlen erzielt, weil er z.B. seinen Mitarbeitern wenig Geld zahlt und folglich eine geringere Marge erhält, bleibt ihm nur die Möglichkeit seine Produkte billig zu produzieren und durch hohe Absatzmengen erfolgreich zu sein (Kostenführerschaft). Andersherum wäre es effektiver teuer zu produzieren (Differenzierung). Die Abteilung die den Mitarbeitern zugewiesen wird beeinflusst ebenfalls drastisch das Spielgeschehen. Durch viele im Sales beschäftigte Mitarbeiter steigt beispielsweise die Chance einen Deal zu gewinnen und die maximale Anzahl an Kunden die gleichzeitig betreut werden können usw. So lässt sich allein durch das Einstellen von Mitarbeitern sein Unternehmen bewusst in eine bestimmte Richtung steuern.

Mit diesem Konzept wird ein Ansatz verfolgt bei dem der Spieler im Verhältnis ein begrenztes Maß an Aktionen ausführen kann, wodurch das Planspiel übersichtlich und intuitiv für den Spieler bleib, aber gleichzeitig durch das Kennzahlen Modell ein hohes Maß an Komplexität und Entscheidungsfreiheit geschaffen wird. Somit kann im Early-Game der Start für den Spieler erleichtert werden und im Mid/End-Game weiterhin erfolgsrelevante Entscheidungen getroffen werden.

2.1.2 Echtzeit & Ranking

Um die Entscheidungen im Spiel noch erfolgsrelevanter gestalten zu können wird im Spiel auf ein ËchtzeitSSystem gesetzt. Die Zeit In-Game basiert auf einem Datumssystem wobei ein Tag 16 Minuten in Real-Time entspricht. Die gesamte Spielzeit des Spiels ist beschränkt auf 10 Geschäftsjahre, dabei ist es irrelevant zu welcher Zeit der Spieler mit dem Spiel beginnt. Sobald sich ein Spieler auf dem Server registriert beginnt seine Spielzeit. Nach den 10 Geschäftsjahren ist das Spiel für den jeweiligen Spieler beendet und sein Unternehmen wird in eine Highscoreliste eingetragen. Somit lässt sich das Spiel unabhängig von anderen Spielern frei spielen, wobei sich der Markt und die Konkurrenzsituation aus allen momentan aktiven Spielern ergibt. Durch das Login System ist die Spieleranzahl beliebig skalierbar. Der Faktor der Zeit spielt im Spiel eine Entscheidende Rolle und hebt den planerischen Aspekt des Spiels deutlicher hervor, hierzu ein Beispiel:

In der Produktion können Maschinen für die Herstellung von Produkten gekauft werden. Diese haben eine maximale Ausbringungsmenge, welche pro Monat angegeben wird: in diesem Beispiel 30.000 Einheiten pro Monat. Die tatsächlich produzierte Menge werden aber pro Tag ausgeschüttet, was einer Menge von 1.000 Einheiten entspricht die dem Warenlager hinzugefügt werden (Ausgegangen von 30 Tagen im Monat). Das Lieferdatum gegenüber aktiven Kunden ist in den Verträgen immer zum Ende des Monats angesetzt. Momentan ist

eine Ausschreibung offen zum 15. des Monats, wobei der Kunde jeweils 50.000 Einheiten pro Monat beziehen will. Der Bestand im Warenlager beläuft sich am 15. des Monats ebenfalls auf 30.000 Einheiten. An dieser Stelle muss der Spieler neben der entstehenden Kosten auch den Faktor der Zeit deutlich in seine Überlegung mit einbeziehen:

- Wie viel produziert meine Maschine in den verbleibenden Tagen bis Monatsende?
- Ist dann durch den Lagerbestand die Bezugsmenge des Kunden gedeckt?
- Kann die Bezugsmenge des Kunden in der verbleibenden Zeit durch eine zusätzliche Maschine gedeckt werden?
- Können die dadurch entstandenen Mehrkosten getragen werden?

Außerdem kann durch den Ansatz einer Echtzeit Simulation zusätzlich das unternehmerische Risiko erhöht werden, wie z.B. Konventionalstrafen bei Nichterfüllung der auszuliefernden Mengen an den Kunden. Spieler die aktiver am Markt sind können sich so ebenfalls einen Zeitvorteil verschaffen, da Spieler die sich eher auf eine Ausschreibung bewerben auch bessere Chancen haben einen Deal für sich zu entscheiden.

2.2 Architektur Markus Böbel

Durch das Echtzeit-Spielkonzept des Planspiels ist es für die Anwendung von Nöten ständig Daten, auszuwerten und zu berechnen. Damit diese Berechnungen erfolgreich durchgeführt werden können, müssen diese unabhängig von der Präsenz der Spieler durchgeführt werden. Dies wird erreicht durch die Einrichtung eines einfachen verteilten Systems, indem die Programmlogik auf einem abgeschlossenen Server liegt, während sich die Spieler lediglich nötige Information anfragen oder ihr Handeln dem Server mitteilt. Wenn nun ein Spieler das Spiel für eine gewisse Zeit unterbricht, können andere Spieler in dieser Zeit das Spiel fortführen.

Als der Software Architektur liegt ein eine einfache Three-Tier-Architektur. (siehe Abbildung 2.1) Wie der Name es vermuten lässt, besteht eine solche Architektur aus drei grundlegenden Elementen.

Datenschicht Die Datenschicht dient der Datenpersistenz und besteht meist aus einer Datenbank.

Logikschicht Diese Schicht bearbeitet die Daten der Datenschicht und bringt diese in einen semantischen Zusammenhang.

Präsentationsschicht Bei dieser Schicht werden die hergerichteten Daten der Logikschicht präsentiert. Sie stellt sozusagen die Schnittstelle zum späteren Spieler da, weil dieser bloß auf der Präsentationsschicht handelt.

Der Grund für diese Schichtentrennung ist die einfachere Austauschbarkeit der verschiedenen Bestandteile. Wenn zum Beispiel das Datenbanksystem erneuert werden soll, so ist dies ohne großen Aufwand möglich. Das Gleiche gilt auch für die anderen Bestandteile.

Im konkreten Falle des Planspiels besteht die Präsentationsschicht aus einer Webseite. Diese sendet die Eingaben des Benutzers weiter an eine REST-Schnittstelle (siehe Kapital 3.2). Diese wird auf einem Tomcat Server initialisiert, auf welchem die Anfragen bearbeitet und das Spielkonzept umgesetzt werden.

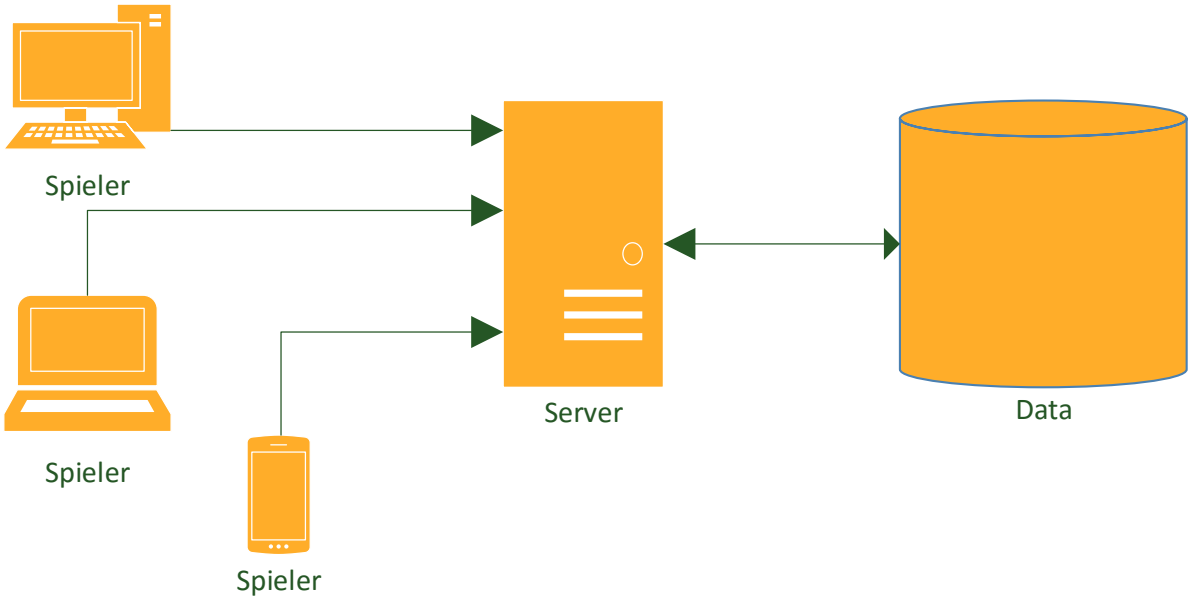

Abbildung 2.1: Aufbau der Applikationsarchitektur

Auf die Datenschicht wird im Falle des Planspiels verzichtet, da es keine Voraussetzung die Daten persistent zu halten und dies einen hohen Mehraufwand darstellen würde. Daraus folgt, dass die Daten nun zur Laufzeit bearbeitet und verwaltet werden. Im Falle eines Serverneustartes oder -ausfalls würden folglich alle Planspieldaten wie zum Beispiel Spielstände verloren gehen. Da aufgrund von Wartungsarbeiten Serverneustarts nichts Ungewöhnliches ist und es sich bei dem Planspiel um ein langwieriges Spiel handelt, ist es von Nöten eine Datenschicht nachträglich zu implementieren.

3 Entwicklung

3.1 Java Backend

Im folgenden wird der Aufbau und die Funktionalität des Java Backends und der einzelnen Klassen dargestellt, auf Grund von Platzmangel wird hier nur auf die wichtigsten Funktionalitäten und Methoden eingegangen.

Die Klasse „Game“ implementiert die Spiellogik und das Echtzeit-Konzept. Hier werden Aktionen, die alle Spieler beziehungsweise Unternehmen betreffen durchgeführt sowie die Unternehmen, beispielsweise für die Festlegung des Gewinners einer Ausschreibung, verglichen.

Die Hauptfunktionalität des Unternehmensplanspiels ist im Package „Unternehmung“ umgesetzt, das, strukturell gesehen, „unter“ der Klasse „Game“ liegt. Hier sind neben der Klasse „Unternehmen“ die einzelnen Abteilungen eines solchen mit ihren verschiedenen Möglichkeiten, die dem Spieler hier zur Verfügung stehen, implementiert (Package „Abteilungen“) sowie das Kennzahlenwesen eines Unternehmens (Package „Kennzahlen“). Darüber hinaus befinden sich in diesem Package die notwendigen Java-Klassen, durch deren Objekte ein objektorientierter Programmieransatz realisiert wird. Einige Methoden werfen Exceptions, die im Package „Exceptions“ abgelegt sind.

3.1.1 Die Klasse „Game“

Luca Dommes

In der Klasse Game ist die Logik und Mechanik des Untenehmensplanspiels implementiert, sie steht, strukturell gesehen, über allen anderen Klassen.

Zunächst einmal ist zu erwähnen, dass es sich hierbei um eine Spezialisierung der Java-Klasse „TimerTask“[1] handelt. Dies dient der Umsetzung des Echtzeit-Ansatzes des Spiels. So wird im Konstruktor der Game-Klasse ein Timer-Objekt erstellt, das mit einem Intervall von 16 Minuten initialisiert wird. 16 Minuten stellen also einen Tag in Spielzeit dar. Im statischen Klassenattribut „gameCalendar“ wird die Spielzeit als Calendar-Objekt[2] dargestellt, das bei jedem Timer Intervall einen Tag hochgezählt wird. Wenn 16 Minuten also ein Spieltag sind vergehen pro Stunde 3,75 Spieltage und dementsprechend an einem Tag (3,75 * 24 =) 90 Spieltage, sprich ein Quartal. Daraus folgt, dass ein Geschäftsjahr 4 (Echtzeit-)Tage dauert.

Die Klasse Game besteht hauptsächlich aus statischen Methoden, um die Zugriffe auf die Klasse zu erleichtern. Wie im Folgenden häufig beschrieben wird oftmals aus den verschiedensten Klassen auf die Klasse Game zugegriffen, unter Anderem um auf das Calendar-Objekt zuzugreifen und somit das aktuelle Datum abzugreifen. Hierfür müsste jeder Klasse das Game-Objekt mitgegeben werden. Durch die statischen Methoden wird dies überflüssig.

In der statischen ArrayList[3] „companies“ sind die Unternehmens-Objekte der einzelnen Mitspieler abgelegt. In einer zweiten ArrayList „companiesArchiv“ werden Unternehmen im Falle des Bankrott abgelegt. Darüber hinaus gibt es noch eine ArrayList „**ausschreibungen**“, was es mit dieser Liste auf sich hat wird in 3.1.2 dargestellt.

In der statischen Map „highscores“ werden die Spielstände gespeichert. Hier wird, entweder im Falle des Spielendes, also zehn Jahre nach Gründung eines Unternehmens, oder beim Bankrott eines Unternehmens, der Eigenkapital-Endbestand sowie der Unternehmensname abgespeichert. An das Frontend wird diese Liste über die Methode „getHighscoresAsTreeMap()“ als TreeMap[4] absteigend sortiert ausgeliefert.

Bei jedem Timer Count, sprich alle 16 Minuten, wird die Methode run() ausgeführt. In dieser Methode wird das Datum (also das Calendar-Objekt) um einen Tag hoch gezählt (updateCounter()). Anschließend wird über die Unternehmens-Liste iteriert und die update()-Methode aller Unternehmen aufgerufen (hierzu mehr in 3.1.2). Basierend hierauf werden die Makrtanteile neu berechnet (siehe folgender Absatz). Zu letzt wird, nur für den Fall, dass der aktuelle Tag der letzte eines Geschäftsjahres, also der 31. Dezember, ist, die Methode updateYearly() aufgerufen, die eine Aufstellung des Jahresabschlusses auslöst (mehr in 3.1.2).

[1] https://docs.oracle.com/javase/8/docs/api/java/util/TimerTask.html
[2] https://docs.oracle.com/javase/8/docs/api/java/util/Calendar.html
[3] https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
[4] https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html

Die Methoden updateMarktanteil(), zur Berechnung der neuen Marktanteile, und die Methode updateAusschreibungen() zur Erteilung eines Zuschlages, dem Löschen alter sowie Generierung neuer Ausschreibungen nehmen Bezug auf die Klassen „Kennzahlensammlung“ (siehe 3.1.2) und „Vertrieb“ (siehe 3.1.2) und werden in den entsprechenden Kapiteln näher erklärt. Der Grund, aus dem sich die Methoden in der Game-Klasse befinden liegt in der Tatsache, dass hierfür übergreifend auf alle Spielstände, sprich Unternehmen, zugegriffen werden muss beziehungsweise übergreifende Vergleiche stattfinden müssen.

3.1.2 Das Package „Unternehmung“

Das Package „Unternehmung“ beinhaltet alle Klassen, die für die Abbildung eines Unternehmens und seiner Funktionen notwendig ist. Es beinhaltet zwei weitere Packages, nämlich „Kennzahlen“, hier sind die notwendigen Java-Klassen zum Kennzahlen-Management (mehr in 3.1.2) untergebracht, und „Abteilungen“, in welchem entsprechend die einzelnen Abteilungs-Klassen eines Unternehmens liegen. Darüber hinaus befindet sich im Package „Unternehmung“ die Unternehmens-Klasse selbst (siehe 3.1.2) sowie (im Package „Objekte“) sämtliche Klassen, deren Objekte für die objektorientierte Programmierung eines Unternehmensplanspiels nützlich sind. All dies wird im nachfolgenden Schritt für Schritt erklärt.

Die Klasse „Unternehmen“

Luca Dommes

Ein Objekt der Klasse „Unternehmen“ repräsentiert ein Unternehmen und somit praktisch auch einen Spieler des Unternehmensplanspiels. Bei der Neuregistrierung eines neuen Spielers wird also ein neues Unternehmen gegründet, sprich ein Objekt der Klasse Unternehmen erzeugt. Dem Konstruktor der Klasse wird der vom Nutzer gewählte Unternehmensname sowie sein Passwort mitgegeben, außerdem das Eigenkapital, sprich Gründungskapital, das allerdings vorgegeben ist. Der Konstruktor kopiert sich das aktuelle Kalenderdatum und hinterlegt dieses als „gruendungsDatum“, auf Grund dessen gleichzeitig ein Datum für das Spielende („gameEnd“) berechnet wird. Dieses ist 10 Jahre nach dem Gründungsdatum. Darüber hinaus richtet der Konstruktor die Kennzahlensammlung (vgl. 3.1.2) ein und führt die Methode initDepartments() aus, die in der Map „abteilungen“ die notwendigen neuen Abteilungs-Objekte des Unternehmens erstellt. Somit ist der erste Schritt getan, das Unternehmen steht.

Neben dem Konstruktor und der Methode zum Initialisieren der Abteilungs-Map beziehungsweise dem Erstellen der neuen Abteilungen gibt es noch die Methode update(), die, aufgerufen bei jedem Timer Count, über die Map der Abteilungen iteriert und für jede Abteilung die update()-Methode (hierzu genaueres in 3.1.2) aufruft sowie gegebenenfalls die updateYearly()-Methode, die beim letzten Timer Count des Jahres aufgerufen wird und den Jahresabschluss (siehe 3.1.2) veranlasst.

Die Klasse „Mitarbeiter“ repräsentiert einen Mitarbeiter des unternehmens und hat die Attribute „name“, „vorname“, „imagelink“ (für das Bild, vgl. 3.1.2), „department“, also die Abteilung in der er arbeitet, „gender“ für das Geschlecht, „gehalt“ und „prodLeistung“, was die mögliche monatliche Produktionsmenge eines einzelnen Mitarbeiters (der in der Abteilung Produktion eingestellt ist) wieder spiegelt.

Umsetzung des Kennzahlenkonzepts

Luca Dommes

Um unternehmensinterne Abläufe möglichst genau abzubilden wird auch auf eine realitätsnahe Implementierung der Buchhaltung von Unternehmen Wert gelegt.

Bilanz und Gewinn- und Verlustrechnung Luca Dommes

Eine wichtige Klasse dieser Buchführung ist die Klasse „Bilanz“. Hier werden Aktiva wie Maschinen, Gebäude, Fertige Erzeugnisse und liquide Mittel sowie Passiva wie Eigenkapital und Fremdkapital, als float-Werte festgehalten und bei den zur Verfügung stehenden Aktionen entsprechend dynamisch angepasst. Für diese dynamische Anpassung wird von Methoden für Aktionen bei denen Zahlungen fließen, etwa der Kauf einer Maschine, über Methoden wie liquiditaetAnpassen() die entsprechenden Werte verändert. Diese Methode wird allerdings auch von Methoden aufgerufen, die zum Beispiel laufende monatlich Kosten „abrechnen“ möchten. Ist hierbei die Liquidität nicht ausreichend kommt es an dieser Stelle zum Wurf einer BankruptException, was für das Unternehmen den Bankrott und somit für den Spieler das Spielende bedeutet.

Darüber hinaus gibt es eine boolesche Methode „liquiditaetAusreichend()“, die zum Beispiel im eben genannten Fall des Kaufs einer Maschine prüft, ob die liquiden Mittel für diese Aktion ausreichend sind und entsprechend, falls dies der Fall ist, true zurück gibt oder alternativ ein „ZuWenigCashException“ wirft, wodurch am Frontend eine passende Nachricht

ausgegeben wird. Als Bilanzsumme werden die beiden Werte „summeAktiva“ und „summePassiva“ herangezogen, die unterjährig entsprechend unausgeglichen sind, jedoch nach dem Jahresabschluss (siehe folgender Absatz) einander gleichen.

Neben der Bilanz ist für die Buchhaltung die Gewinn- und Verlustrechnung, kurz GuV, ebenso wichtig. Hierfür existiert eine weitere Java-Klasse. Hier werden Aufwendungen für Rohstoffe, Werbung, Gehälter, Energie, Soziale Leistungen, Zinsen, Fremdinstandhaltung und Schadensersatz auf der einen sowie Umsatzerlöse auf der anderen Seite dokumentiert. Für die dynamische Anpassung der relevanten Werte wird bei jedem Timer Count durch die update()-Methode der Klasse „Kennzahlensammlung“ zum Einen die Methode „importAufwandUndErlös()“ aufgerufen, die die entsprechenden Aufwendungen und Erlöse um die täglich (also pro Timer Count) konsumierten beziehungsweise umgesetzten Werte der Einzelnen Abteilungen erhöht, und zum Anderen die Methode „getTaegliche Liquiditaetsveraenderung()“, die wiederum von der Bilanz-Klasse genutzt wird, um die täglichen Ein- und Auszahlungen auch im Bilanzposten „liquide Mittel“ (über liquiditaetAnpassen()) zu erfassen. Um dem Spieler einen Überblick über die Entwicklung seiner Ein- und Ausgaben zu ermöglichen werden die monatlichen Aufwendungen und Erlöse in den float-Werten „aufwendungenArchiv“ und „erloeseArchiv“ kumuliert und schließlich in der LinkedList[5] „archiv“ über die am Monatsende aufgerufene Methode „archivieren()“ festgehalten. Hierzu wird in archivieren() ein neues GuV-Objekt mit einem Spezialkonstruktor, der die Werte aufwendungenArchiv und erloeseArchiv kopiert, erstellt, das anschließend, versehen mit dem aktuellen Datum, in zuvor genannter LinkedList abgelegt wird. Die Attribute aufwendungenArchiv und erloeseArchiv der „Unternehmens-GuV“ werden wieder entsprechend zurück gesetzt, um die Werte des neuen Monats zu sammeln.

Über die updateYearly()-Methode der Unternehmens-Klasse wird am Jahresende die GuV-Methode „jahresAbschluss()“ aufgerufen. Diese berechnet aus allen Aufwendungen und Erlösen den „jahresUeberschuss“, in Höhe dessen anschließend das Eigenkapital in der Bilanz (siehe „eigenkapitalAnpassen()“) angepasst wird, sodass die Bilanz wieder ausgeglichen ist. Die Werte der Aufwendungen und Erlöse werden wieder auf null gesetzt, die nächste Periode kann beginnen.

Weitere Kennzahlen und die Kennzahlensammlung Luca Dommes

Mit eine der wichtigsten Klassen des Java Backends ist die Klasse „Kennzahlensammlung“. Hier laufen, wie der Name bereits zu vermuten lässt, sämtliche Kennzahlen des Unternehmens zusammen. Hier können also zentralisiert alle Berechnungen oder Aktualisierungen

[5] https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html

der Kennzahlen durchgeführt werden. Einige Abteilungs-Klassen greifen auf die Kennzahlensammlung zu, etwa um anhand einer oder mehrerer bestimmter Kennzahlen Entscheidungen zu treffen oder aber selbige zu verändern. Aus diesem Grund wird das Objekt der Kennzahlensammlung eines Unternehmens auch so gut wie jeder weiteren Klasse, in der Regel im Konstruktor, weiter gegeben, sodass die jeweilige Klasse damit „arbeiten“ kann.

Dem Konstruktor der Kennzahlensammlung wird das Unternehmen mitgegeben. Dieses wird weitergegeben an die GuV sowie an die Bilanz, die zwei weitere Klassenattribute darstellen, die im Konstruktor erstellt werden und somit die Buchhaltung des Unternehmens darstellen. Darüber hinaus gibt es ein Attribut „Marktanteil“ für den absoluten mengenmäßigen Marktanteil eines jeden Unternehmens und eine Map „weicheKennzahlen“, die weitere Kennzahlen enthält auf die nachfolgend eingegangen wird. Das Attribut „maxNeueMitarbeiter“ ist die Anzahl neuer Mitarbeiter, die maximal eingestellt werden können, hierzu jedoch mehr in 3.1.2.

Bei den sogenannten weichen Kennzahlen handelt es sich um nicht-monetäre Kennzahlen. Diese sind Mitarbeiterzufriedenheit, Kundenzufriedenheit, Image, Bekanntheitsgrad und Verkaufswahrscheinlichkeit. Jede dieser Kennzahlen ist als Subklasse der Superklasse „Kennzahl“ implementiert. Ein Objekt der Klasse Kennzahl besteht aus Basiswert („basiswert“), einem Modifier („modifier“) sowie einem Endwert („wert“). Der „wert“ setzt sich zunächst aus Basiswert und Modifier zusammen. Über die Methode „getWert()“ wird der Endwert zurück gegeben, berechnet mit einem Tangens Hyperbolicus, der sich der eins nur nähert, sie jedoch nie trifft beziehungsweise überschreitet, schließlich können und sollen die hier definierten Kennzahlen nicht mehr als 100 Prozent ergeben. Dass der Wert, wenn er bereits sehr hoch ist (80% +) langsamer steigt ist ein weiterer gewünschter Nebeneffekt, da es zum Beispiel schwerer sein soll die Mitarbeiterzufriedenheit von 80 auf 90 Prozent zu steigern, als von zehn auf 20 Prozent. Des Weiteren ist die Idee dahinter, dass der Basiswert abhängig von bestimmten Zahlen des Unternehmens ist und durch den Modifier modifiziert wird. Am Beispiel der Mitarbeiterzufriedenheit bedeutet dies, dass sich der Basiswert aus dem durschschnittlichen Gehalt geteilt durch einen festgelegten Wert berechnet, während der Modifier auf die Höhe insofern Einfluss nimmt, als dass er einerseits beispielsweise durch die Auszahlung betrieblicher Sonderzahlungen hoch gesetzt wird, andererseits allerdings auch wieder mit der Zeit, also jeden Timer Count um einen gewissen Wert, wieder sinkt. Dieses Prinzip gilt in gleicher Art und Weise auch für die Kundenzufriedenheit und den Bekanntheitsgrad. Der Basiswert des Images berechnet sich aus der Mitarbeiter- und Kundenzufriedenheit und die Verkaufswahrscheinlichkeit (siehe 3.1.2), die für eine hohe Vertragsabschlussquote entscheidend ist, setzt sich wiederum zusammen aus Image und Bekanntheitsgrad.

Die Marktanteile der Unternehmen werden täglich, also bei jedem Timer Count, neu berechnet. Dies geschieht über die Methode updateMarktanteile() in der Game-Klasse, da hierzu auf die Absatzzahlen aller Unternehmen zugegriffen werden muss. Hier wird zunächst die Methode getGesamtabsatz() aufgerufen, die über die Unternehmens-Liste iteriert und für jedes Unternehmen die Anzahl der im vergangen Monat abgesetzten Produkte abfragt (diese Zahlen werden aus der Klasse beziehungsweise Abteilung „Vertrieb“ gewonnen (siehe 3.1.2)) und die Summe dieser, also den gesamten Absatz des Marktes, zurück gibt. Anschließend iteriert die updateMarktanteile()-Methode erneut über die Unternehmen, setzt die abgesetzte Menge eines jeden Unternehmens ins Verhältnis zum soeben ermittelten Gesamtabsatz und setzt die somit gewonnene Kennzahl des absoluten mengenmäßigen Marktanteils in der Klasse Kennzahlensammlung (siehe 3.1.2).

Die update()-Methode dieser Klasse, die bei jedem Timer Count aufgerufen wird, ruft wiederum die Methode „kennzahlenRuntersetzen()“ auf, die den Modifier aller weichen Kennzahlen etwas herab setzt, sofern dieser nicht schon bei null liegt. Auch dies spiegelt die Realität des Wettbewerbs wieder, in dem es sich kein Unternehmen leisten kann sich „auszuruhen“. Des Weiteren ruft die update()-Methode die Methode „berechnen()“ auf, die über alle weichen Kennzahlen iteriert und eine Neuberechnung veranlasst. Anschließend wird die Liquidität der Bilanz durch die aus der GuV gewonnen Liquiditätsveränderungen angepasst (vgl. 3.1.2).

Ähnlich wie es in der GuV umgesetzt ist werden gewisse Daten auch hier archiviert, allerdings nicht monatlich sondern jährlich. Hierbei handelt es sich um den Jahresabschluss, sprich die GuV zum Jahresende sowie die Schlussbilanz. Hierfür wird die archivieren()-Methode der Kennzahlensammlung von der updateYearly()-Methode der Unternehmens-Klasse aufgerufen. Die archivieren()-Methode erstellt, ähnlich wie es in der GuV der Fall ist, ein neues Kennzahlensammlungs-Objekt anhand eines kopierenden Konstruktors, der die gesamte Bilanz sowie GuV kopiert. Dieses Objekt wird, kombiniert mit dem Datum, in der Map „archiv“ abgelegt.

Die einzelnen Abteilungen und ihre Funktionen

Die einzelnen Abteilungen sind das, was ein Unternehmen ausmacht. Das Grundgerüst einer Abteilung ist durch die Klasse „Abteilung“ definiert, die jeweils für jede einzelne Abteilung im Package „Abteilungen“ spezialisiert wird. Die Superklasse „Abteilung“ hat als wichtigste Attribute einen Namen als String, eine ArrayList mit Mitarbeiter, die der jeweiligen Abteilung zugeordnet sind, sowie einen float-Wert „aktKosten“, der die aktuellen laufenden Kosten der Abteilung widerspiegelt. Diese Kosten werden je nach Abteilung individuell

gesetzt, über die bei Timer Counts durch die jeweilige update()-Methode aufgerufenen Methoden „getKosten()“ und „getMitarbeiterKosten()“ werden die täglich anfallenden Kosten sowie monatlich anfallenden Gehälter an die GuV und die Bilanz zur „Verbuchung“ weiter gegeben.

In der Abteilungs-Klasse ist auch die Methode „addMitarbeiter()“ implementiert. Die semantische Zugehörigkeit dieser Methode liegt eigentlich in der Abteilung „Human Resources“, die Methode ist jedoch hier implementiert, sodass der neu eingestellte Mitarbeiter direkt einer entsprechenden Abteilung zugeordnet ist und in der ArrayList „mitarbeiter“ der Abteilung aufgenommen wird. Am Frontend wird die Funktion, Mitarbeiter einzustellen, dementsprechend in der Abteilung „Human Resources“ angezeigt. Der genaue Vorgang des Einstellens eines Mitarbeiters wird im folgenden Kapitel 3.1.2 näher dargelegt.

Human Resources Luca Dommes

In der Abteilung „Human Resources“, kurz HR, ist die weitere Verwaltung der Mitarbeiter implementiert. So hat man am Frontend hier entsprechend die Möglichkeit, wie bereits erwähnt, Mitarbeiter einzustellen, aber auch diesen zu kündigen beziehungsweise sich die Liste der Mitarbeiter anzusehen.

Zum Einstellen eines neuen Mitarbeiters wird (in der Klasse „Abteilung“) die Methode „addMitarbeiter()“ aufgerufen. Hier kommt nun auch der Integer-Wert „maxNeueMitarbeiter“ in der Kennzahlensammlung (vgl. 3.1.2) zum Einsatz. Es ist nämlich nur möglich Mitarbeiter in anderen Abteilungen als Human Resources einzustellen, wenn genügend Mitarbeiter in Human Resources, People Manager also, eingestellt sind. Der Startwert von „maxNeueMitarbeiter“ liegt bei 10. Das bedeutet, dass die ersten zehn Mitarbeiter einfach so eingestellt werden können, erst ab dem elften Mitarbeiter ist das Einstellen eines Managers in HR notwendig. Wird ein neu einzustellende Mitarbeiter in Human Resources eingestellt erhöht sich entsprechend der Wert „maxNeueMitarbeiter“ um neun, da ein HR-Mitarbeiter für zehn andere Mitarbeiter zuständig ist. Soll einer oder mehrere Mitarbeiter in einer anderen Abteilung als HR eingestellt werden, so ist dies nur möglich, wenn maxNeueMitarbeiter in der Kennzahlensammlung größer oder gleich der Anzahl neu einzustellender Mitarbeiter ist, maxNeueMitarbeiter wird anschließend entsprechend um diese Anzahl korrigiert. Sind nicht genügend People Manager in HR vorhanden, maxNeueMitarbeiter liegt also unter der Anzahl der vom Benutzer gewünschten neu Einzustellenden, so wird eine „ZuWenigMitarbeiter-Exception“ geworfen, die am Frontend eine entsprechende Nachricht auslöst, dass zunächst einer oder mehrere Mitarbeiter in HR angestellt werden müssen, um diese Einstellung(en) zu ermöglichen, und der Vorgang wird abgebrochen. Zur Einstellung neuer Mitarbeiter wird die

open-source API „Random User Generator“[6] verwendet, die zufällige Personen generiert. Für die Zwecke des Unternehmensplanspiels wird von dieser API ein Name, Bild und Geschlecht des neuen Mitarbeiters abgefragt beziehungsweise generiert. Diese Daten werden über die URL „https://randomuser.me/api/?results=“ + anzahl + „&inc=name,picture,gender“ abgefragt, durch einen BufferedReader[7] in Verbindung eines InputStreamReaders[8] eingelesen und anschließend als JsonObject[9] in einem JsonArray[10] zwischen gespeichert. Von hieraus können anschließend neue Mitarbeiter-Objekte erstellt werden und mit den entsprechenden Daten aus dem JsonArray gefüllt werden. Das Mitarbeiter-Gehalt wird der Methode übergeben und entsprechend gesetzt, da diese Entscheidung durch den Spieler fällt.

In der HR-Abteilung gibt es darüber hinaus noch die Möglichkeit seinen Mitarbeitern soziale Leistungen anzubieten. Hierfür sind zwei weitere Klassen implementiert: „SozialProjekt“ und „ZeitGeld“, was eine Spezialisierung von SozialProjekt ist. Ein SozialProjekt ist beispielsweise die Inbetriebnahme einer Kantine, bei ZeitGeld handelt es sich um freiwillige Sonderzahlungen des Unternehmens wie zum Beispiel Weihnachtsgled. Diese Maßnahmen können gestartet und nach belieben wieder gestoppt werden (Methoden start() und stop()), verursachen, wenn sie aktiv sind (boolesche Variable „active“), einmalige und/oder laufende Kosten und erhöhen die Mitarbeiterzufriedenheit (siehe start(), stop() und update()). Diese Projekte werden in der Klasse HR in einer ArrayList „projekte“ verwaltet.

Produktion Luca Dommes

In der Abteilung „Produktion“ findet die Leistungserstellung statt. Bevor jedoch produziert werden kann müssen einige Voraussetzungen erfüllt werden.

So ist für die Produktion sowie für die Lagerung fertiggestellter Produkte der Faktor Boden notwendig. Hierfür wurde eine neue Java-Klasse namens „Halle“ implementiert. Diese Halle repräsentiert einerseits eine Produktionshalle, die eine gewisse Kapazität, also Anzahl von Stellplätzen, für Maschinen hat und andererseits eine Lagerhalle, die eine gewisse Lagerkapazität (= maximal lagerbare Produkte) hat. Eine Halle hat neben der Art (also „Produktionshalle“ oder „Lagerhalle“) und der Kapazität die Attribute „groesse“ und „preis“. Die Größe („groesse“) ist ein Integer-Wert, der eins, zwei oder drei sein kann und vom Spieler gewählt wird. Abhängig von der Art der Halle Größe wird dann in der Methode „findPreisundKapazitaet()“ der Preis und die Kapazität festgelegt. Erzeugt werden diese

[6] https://randomuser.me/
[7] https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html
[8] https://docs.oracle.com/javase/8/docs/api/java/io/InputStreamReader.html
[9] https://static.javadoc.io/com.google.code.gson/gson/2.8.0/com/google/gson/JsonObject.html
[10] https://static.javadoc.io/com.google.code.gson/gson/2.8.0/com/google/gson/JsonArray.html

Objekte durch die Methoden „produktionshalleKaufen()“ und „lagerhalleKaufen()“ in der Produktions-Klasse, die ebenso die entsprechenden Bilanzposten anpassen.

Neben dem Boden sind für die Produktion ebenso Maschinen notwendig. Hierfür gibt es die Klasse „Maschine“. Das Attribut „klasse“ ist vergleichbar mit der Größe der Halle, denn hiervon hängt die Höhe der Attribute „kapazitaet“ (= Ausbringungsmenge pro Monat) und „anschaffungskosten“ ab, diese werden gesetzt in der Methode „findKapazitaetUndAnschaffungskosten()“. Weitere Attribute sind „produkt“, da eine Maschine nur ein bestimmtes Produkt produzieren kann, „energiekosten“ und „status“. Die Energiekosten sind bei allen Maschinen gleich, sodass sich hierdurch die größere Maschine lohnt. Der Status spiegelt die Abnutzung wieder, er wird also bei jedem Timer Count herunter gesetzt, gleichzeitig steigen die Energiekosten (siehe „statusUndEnergiekstRuntersetzen()“). Diesem Effekt kann durch Reparaturen entgegengewirkt werden (siehe „reparieren()“). In der Produktions-Klasse werden die Maschinen in der ArrayList „maschinen“ verwaltet. Maschinen können auch wieder verkauft werden, der Wiederverkaufspreis richtet sich hier nach den halben Anschaffungskosten multipliziert mit dem Status der Maschine (siehe „maschineVerkaufen()“).

Ist die letzte Voraussetzung zur Produktion, nämlich der Faktor Arbeit erfüllt, sprich gibt es in der Abteilung Produktion Mitarbeiter, so kann produziert werden (siehe „produzieren()“), falls nicht wirft diese Methode eine entsprechende Exception. Dieser Methode gibt der Nutzer Informationen zum „Produkt“ beziehungsweise der „Produktlinie“ mit. Dies sind zwei weitere Java-Klassen. Ein Produkt besteht aus den Attributen „name“ (etwa „Rucksack“ oder „Duffel“), „qualiteatsstufe“ (A, B oder C), „herstellkosten“ und „forschungsbonus“ (vgl. Forschung). Abhängig von dem Produkttyp und der Qualitätstufe (also zum Beispiel „Rucksack der Qualitätsstufe A“) werden die Herstellkosten für das Produkt gesetzt (siehe „findHerstellkosten()“). Die Klasse „Produktionslinie“ enthält neben einem solchen Produkt-Objekt Informationen über Menge, Laufzeit, Beginn, Ende und ID (zusammengesetzt aus Produkttyp und Qualitätsstufe). Somit können Objekte der Klasse Produktlinie einerseits als Produktionsauftrag (siehe ArrayList „aufträge“), und andererseits auch als Objekt für die Lagerbestände (siehe ArrayList „lager“) verwendet werden.

In der Methode „produkteFertigstellen()“, die über Produktion.update() aufgerufen wird, wird über „aufträge“ iteriert und für jedes Produktlinien-Objekt aber auch in Hinblick auf beispielsweise die Produktionskapazität eine umfangreiche Prüfung durchgeführt, etwa ob (immer noch) genügend Maschinen, Mitarbeiter oder Lagerplatz vorhanden sind. Falls dies nicht der Fall ist wird eine entsprechende Exception geworfen, die dem Nutzer am Frontend das Problem anzeigt. Die Produkte können hierdurch nicht produziert werden beziehungsweise werden „entsorgt“ wenn nicht genügend Lagerplatz vorhanden ist. Die maximale Menge

der zu produzierenden Produkte richtet sich nach sowohl Mitarbeiter-, als auch der Maschinenkapazität und orientiert sich am jeweils niedrigeren Wert (siehe Verwendung der Methoden getMaxMitarbeiterProdMenge(), getMaxMaschProdMengen(), getMaxMaschProdMengeByProdukt(), getVerfuegbareMengeByProdukt). Stimmen alle Parameter so werden die fertigen Erzeugnisse in der ArrayList „lager“ abgelegt und auch die entsprechenden Bilanzposten angepasst.

Marketing Jan Scheuermann Die Abteilung Marketing dient der Erhöhung der Verkaufswahrscheinlichkeiten und des Bekanntheitsgrades. Nachdem hier Mitarbeiter hinzugefügt worden sind, kann man eine Marketingkampagne, oder aber eine Marketingforschung starten, indem man eine Laufzeit festlegt, wie viele Tage das Projekt andauern soll. Was diese genau machen wird in den zugehörigen Unterkapiteln genauer erläutert.

Hat man eine Marketingkampagne, oder -forschung mit genügend freien Mitarbeitern erfolgreich begonnen, wird mit jedem Aufruf der update() Methode der Bonus aller laufenden Marketingkampagnen/-forschungen durch die Methoden updateMafos() und updateMarketingkampagnen() der jeweiligen Kennzahl hinzugefügt. Die laufenden Kosten werden als Aufwendungen beachtet und es wird überprüft, ob das jeweilige Enddatum mit dem aktuellen Datum übereinstimmt. Sind die Daten gleich, wird das laufende Projekt entfernt.

Die noch verfügbaren Mitarbeiter lassen sich durch die Methode getVerfuegbareMitarbeiter ausgeben, indem alle Mitarbeiter die Kampagnen zugeteilt sind, mit denen, welche in Marktforschungen aktiv sind, zusammengezählt werden. Dieser errechnete Wert wird dann mit den in der Marketingabteilung angestellten Mitarbeitern subtrahiert, und der Restwert schließlich ausgegeben.

Marketingkampagne Bei den Marketingkampagnen gibt es die Möglichkeit, aus vier verschiedenen Szenarien zu wählen, die sich in ihren Kosten, den benötigten Mitarbeitern und natürlich der Auswirkung hin unterscheiden.

Mögliche Kampagnen sind Social Media, Print, Radio und TV. Die laufenden Kosten und die Anzahl der benötigten Mitarbeiter liegen je nach Kampagne zwischen 1.000 und einem Mitarbeiter für Social Media, und 130.000 und sieben Mitarbeiter für den TV-Spot. Der Bonus für den Bekanntheitsgrad nimmt natürlich mit Anstieg der Kosten und benötigten Mitarbeitern zu. Sollten für die Kampagnenaufnahme nicht genügend Mitarbeiter vorhanden sein, wird die Exception ZuWenigMitarbeiter geworfen. Analog wird bei zu geringem Geldvermögen die Exception ZuWenigCash ausgegeben.

Die Idee ist, dass zu Beginn einer Marketingkampagne die laufenden Kosten mit der Laufzeit multipliziert werden. Nach der erfolgreichen Überprüfung, ob man sich die Kampagne leisten

kann, werden die Kosten zu Aufwendungen und die Kampagne beginnt, tägliche Vorteile zu erbringen.

Der Hintergedanke hier ist, dass durch Publicity der Bekanntheitsgrad des Unternehmens erhöht wird. Je höher die Kosten desto mehr Menschen werden durch die Werbung erreicht und somit wird der Effekt größer. Leider kann man den Umfang einer bestimmten Kampagne nicht verändern. Man ist also gezwungen, falls man TV-Werbung schalten möchte, einen festen Betrag zu zahlen, wie er beispielsweise bei einer Werbeunterbrechung in „Wer wird Millionär“ anfallen würde. Ist einem dies zu teuer, können ohne Bedenken günstigere Varianten gewählt werden. Die erzielbaren Effekte bleiben bei jeder Kampagne gleich, man ist also nicht gezwungen nach einer Weile die Kampagne zu wechseln, weil die Rentabilität gesunken ist.

Man kann so viele Marketingkampagnen starten, wie Geld und Mitarbeiter vorhanden sind, selbst wenn es mehrere desselben Typs sind.

Marktforschung Bei der Marktforschung hat man die Wahl zwischen drei Möglichkeiten. Welche genau gestartet wird, legt man durch Angabe des Umfangs fest. Wenn eine Marketingforschung gestartet werden soll, wird zuerst überprüft, ob ein Forschung desselben Umfangs bereits am Laufen ist. Trifft dies zu, wird die Exception LaeuftBereits geworfen. Auch wird kontrolliert, ob ausreichend freie Mitarbeiter und genügend liquide Mittel vorhanden sind. Sind diese Vorgaben nicht erfüllt, werden die Exceptions ZuWenigMitarbeiter oder ZuWenigCash geworfen. Eine Laufzeit kann man hier nicht angeben, da diese bereits für die verschiedenen Typen festgelegt ist. Die einzelnen Marktforschungen unterscheiden sich in Hinsicht der Dauer, der Anzahl an benötigten Mitarbeitern und dem täglichen Bonus für die Verkaufswahrscheinlichkeit. Marktforschungen können so drei, sechs oder neun Monate andauern und jederzeit abgebrochen werden. Marktforschungen kosten täglich 50 , zzgl. Der Gehälter der eingesetzten Mitarbeiter und erbringen einen Bonus zwischen 0,05 und 0,3.

Die Idee ist, dass bei einer Marktforschung Mitarbeiter nach draußen geschickt werden, um Passanten und andere Personen zu befragen. Je länger die Marktforschung also andauert, desto mehr Wissen und Informationen können gesammelt werden und desto besser wird das Bild davon, was die Kunden eigentlich haben wollen. Dementsprechend wird nach Abschluss einer Marktforschung die Verkaufswahrscheinlichkeit des Unternehmens erhöht.

Aus zeitlichen Gründen konnten die Marktforschungen allerdings leider nicht mit dem Frontend verknüpft werden. Eine Implementierung besteht deshalb rein auf Backend-Seite.

Vertrieb Jan Oehlers

Forschung Jan Scheuermann In der Abteilung Forschung können verschiedene Boni für den Spieler freigeschaltet werden. Sind hier Mitarbeiter eingestellt, kann man mit diesen ein Forschungsprojekt starten. Nach der Auswahl erhält man die Möglichkeit, zwischen einer Herstellkostenreduzierung des Produktes, oder der Verbesserung der Kundenzufriedenheit. Hintergedanke hier ist, dass man dies durch Optimierungen im Herstellungsprozess, sei es durch herausgefundene Best-Practices, durch Materialreduzierung, oder aber durch Reduzierung des Ausschusses, erreichen kann. Der Grund dafür, die Kundenzufriedenheit durch Forschungen zu erhöhen beruht darin, dass die Qualität der Produkte bei gleichbleibenden Kosten erhöht wird. Dies kann beispielsweise durch ein neues Design, neue Funktionalitäten oder bessere Produktions-Prüfungsverfahren, bei denen minderwertige Produkte eher erkannt werden, erreicht werden. Die maximale Wert für die Herstellungskostenreduzierung eines Produktes beträgt 25 % und für den Kundenzufriedenheitsbonus je Produkt 8.3 %. Würde der neue Wert höher als der Maximalwert sein, wird in den Methoden setImagebonus() und setForschungsbonus() der Wert auf den Maximalwert gesetzt und jeglicher Restwert verworfen.

Die Liste mit den Produkten und den Forschungsboni für die Kundenzufriedenheit sind in der Forschungsklasse abgelegt. Bei jedem Erhöhen des Wertes durch das Abschließen von Forschungsprojekten wird zusätzlich in den Methoden setImagebonus() und setForschungsbonus() überprüft, ob der neue Wert den Maximalwert überschreiten würde. Falls dies zutreffend ist, wird der Wert auf den Maximalwert gesetzt und jeglicher Restwert verworfen. Analog wird bei den Herstellungskosten vorgegangen. Hier ist die Werteliste jedoch in der Produktionsklasse abgelegt, da die Kostenreduzierung bei den Herstellkosten mit jedem Produktionstag abgefragt wird. Dies bietet einen Performance Vorteil, da keine Abfragen auf ein Forschungsabteilungsobjekt stattfinden müssen. Möchte man jetzt ein Forschungsprojekt mithilfe der Methode forschungsprojektStarten() beginnen, müssen verschiedene Angaben getätigt werden. Zum einen die Anzahl der zugeteilten Mitarbeiter, die zeitliche Dauer des Projekts, ob Herstellkosten reduziert werden sollen oder die Kundenzufriedenheit erhöht, und natürlich das zu beforschende Produkt. Ob genügend freie Angestellte vorhanden sind wird über die Variable beschäftigteMitarbeiter kontrolliert, die die Anzahl an Mitarbeitern wiedergibt, welche momentan laufenden Forschungsprojekten zugeteilt sind. Falls hier nicht genügend freie Mitarbeiter angestellt sein sollten, wird die Exception ZuWenigMitarbeiter geworfen. Zusätzliche Kosten entstehen bei der Aufnahme von Projekten nicht, lediglich das Gehalt der angestellten Forscher bestimmt die Ausgaben der Forschungsabteilung.

Mit jedem vergangen Tag wird dann über die update() Methode geprüft, ob das Enddatum eines Projektes mit dem aktuellen Tag übereinstimmt. Ist dies zutreffend, wird das Projekt durch Aufruf der Methode forschungsprojektAbschließen() erfolgreich abgeschlos-

sen. Der Wert berechnet sich dann aus der Mitarbeiteranzahl, den vergangenen Tagen und der Mitarbeiterzufriedenheit im Unternehmen, multipliziert mit einem festgelegten Faktor, der die Arbeitsleistung passend zu dem Bonus umwandelt. Will man Mitarbeiter entlassen, oder anderen Forschungsprojekten zuteilen, gibt es die Möglichkeit, laufende Forschungsprojekte mithilfe der Methode forschungsprojektAbbrechen() vorzeitig abzubrechen. Der Bonus errechnet sich dann aus der bereits vergangenen Zeit und einem zusätzlichen Straffaktor von 0.7, welcher davon herrührt, dass die Forscher vor regulärem Ende das Projekt gezwungenermaßen abschließen mussten und so teilweise Ideen oder Ansätze nicht vollständig umsetzen konnten. Mitarbeiter, welche den Projekten zugeteilt gewesen sind, werden dann natürlich wieder frei um entweder in anderen Forschungsprojekten arbeiten zu können, oder aber um ohne Auswirkungen aus dem Unternehmen entlassen zu werden. Entlässt man allerdings Mitarbeiter aus der Forschungsabteilung, wenn alle Angestellte Projekten zugeteilt sind, wird dem zuletzt hinzugefügten Forschungsprojekt einer der Mitarbeiter entnommen, dessen Leistungen ebenfalls nichtig gemacht werden. Sinkt so die Mitarbeiteranzahl eines Projekts auf null, wird das Projekt ohne jegliche Auswirkungen eingestampft. Des Weiteren ist die Forschung der einzige Weg, um die Kundenzufriedenheit zu erhöhen. Deswegen sollte hiervon unbedingt Gebrauch gemacht werden.

Anzumerken ist noch, dass mit Produkten, an welchen bereits geforscht wird, keine weiteren Forschungen gestartet werden können. Hierfür gibt es die beiden Listen verfügbareProdukte und beforschteProdukte. Das hat den einfachen Hintergrund, dass mit Forschungsbeginn ein Tranformationsprozess des Produktes eingeleitet wird, nach welchem Veränderungen stattfinden. Zwei parallellaufende Projekte zu dem gleichen Produkt würden also nach Abschluss theoretisch zwei unterschiedliche Zustände des Produkts erzeugen.

Finanzen Jan Scheuermann Die Abteilung Finanzen dient dem Aufnehmen von Krediten zur Finanzierung von Investitionen. Sobald man hier einen Mitarbeiter angestellt hat, kann damit begonnen werden Geld für das Unternehmen zu leihen. Zur Aufnahme eines Kredits werden noch weitere Angaben benötigt, so z.B. wie hoch der Betrag ist, und wie lange der Kredit gewährt werden soll. Der genaue Zinssatz für den Kredit wird dann durch Errechnung des neuen Verschuldungsgrades nach Abschluss festgelegt. Beginnend bei einem Satz von 2 % steigert sich der Zinssatz dann bis zu maximal 6 %, da sich mit Anstieg des Verschuldungsgrades auch das Kreditrisiko erhöht. Jeder Verschuldungsgrad oberhalb von 200 % verwehrt die weitere Kreditaufnahme und wirft die Exception ZuHochVerschuldet. Kreditinstitute empfinden das Unternehmen dann nicht mehr als kreditwürdig, da die Wahrscheinlichkeit einer Insolvenz relativ hoch ist. Auch wird überprüft, ob der Kredit noch vor Spielende (Also innerhalb der Lebensdauer des Unternehmens) abbezahlt werden würde. Tut er dies nicht, wird die Exception LaufzeitZuHoch geworfen.

Kredite besitzen abgesehen von der Höhe, der Laufzeit und dem errechneten Zinssatz noch einen Wert für die Tilgung, welcher sich aus der Höhe geteilt durch die Laufzeit ergibt, einen für die Zinsen, welche sich aus Zinssatz multipliziert mit dem Restwert ergibt, und einen letzten für die Annuität, die sich aus Addition von Zinsen und Tilgung ergibt. Die Annuität ist schließlich der Betrag, welcher von unseren Liquiden Mitteln gedeckt werden muss, da andernfalls das Unternehmen durch Aufruf der Exception Bankrupt aufhört zu existieren.

Mit jedem Aufruf der update() Methode wird nun überprüft, ob der Kredit fällig ist und ob die Laufzeit zu Ende ist. Sollte ein ganzes Jahr seit Kredit Aufnahme vergangen sein, wird der jährliche Zinssatz berechnet und zusammen mit der Tilgung zur Annutität. Ist die Laufzeit schließlich auf null gesunken, wird überprüft, wie viele Monate seit der letzten Annuitätenzahlung vergangen sind. Der herauskommende Wert wird dann als Faktor verwendet, um die Restzinsen und die Resttilgung für den Kredit, sprich die Restannuität, berechnen zu können. Dafür werden die Werte dann, falls sie Aufwendungen darstellen, an die GuV weitergegeben, und ansonsten in die Bilanz übernommen. Hier werden dann die liquiden Mittel und das Fremdkapital entsprechend angepasst. Sollten nicht genügend liquide Mittel vorhanden sein, wird ebenfalls die Exception Bankrupt geworfen.

3.2 Rest-Schnittstelle Markus Böbel

Die Verbindung zwischen der Programmlogik im Java Backend und dem Angular-2-Frontend wird über eine so genannte Representational State Transfer (REST)- Schnittstelle geschaffen. Dieses basiert auf dem Hyper Text Transfer (HTTP), welche es ermöglicht verschiedene Daten zwischen einem Client und einem Server zu senden. Eine Nachricht des Hyper Text Transfer besteht aus einem Header, welcher Meta-Daten einer Anfrage, und einen Body, welcher Daten zur Übertragung enthält. Basierend auf dieser Struktur können nun vom Client aus Anfragen an bestimmte Server URLs gesendet werden. Als Grundlage besagter Anfragen stehen 9 Verben (`GET`,`POST`,`PUT`,`DELETE`,`PATCH`,`HEAD`,`OPTIONS`,`CONNECT`,`TRACE`), welche die Art der Anfrage beschreibt. Im Folgenden werden die häufigsten 4 Anfragen gezeigt:

GET Mit einer `GET`- Anfrage werden reine Informationen angefragt. Daten im Body sind nicht erlaubt.

POST Die `POST`-Anfrage wird dazu verwendet, um neue Elemente auf dem Server zu erstellen. Das neue Element kann im Body der Anfrage in Form eines JSON Textes übertragen werden

PUT Während `POST` neue Objekte auf dem Server erstellt, dient eine `PUT` Anfrage dazu, Daten zu updaten. Das geupdatete Objekt wird im Body der Anfrage mitgeliefert.

DELETE Wie der Name vermuten lässt, dient die `DELETE`-Anfrage dazu Objekte zu löschen.

Um konkrete Elemente anzusprechen, können Parameter innerhalb des URLs übergeben werden. So wird das Unternehmen mit der ID 1 zum Beispiel über die URL `localhost:8080/rest/compani` angesprochen. Aufgrund dieser verschiedenen Anfragen kommt es zu unterschiedlichen Ergebnissen beim Aufruf einer bestimmten URL. Wird zum Beispiel die URL `/rest/companies/1` aufgerufen kommt es zu folgenden Ergebnissen:

GET Es wird das erste Unternehmen zurückgegeben.

POST Es wird ein neues Unternehmen erstellt, welches nun die ID 1 hat.

PUT Es wird das Unternehmen mit der ID 1 überarbeitet.

DELETE Das Unternehmen mit der ID 1 wird entfernt.

Nachdem eine Anfrage erfolgreich den Server erreicht hat, bearbeitet dieser die besagten Anfragen unter Berücksichtigung der oben genannten Verben. Das Ergebnis der Requests wird anschließend an den Client gesendet. Anbei befindet sich eine Nummer, welche den Status der Anfrage dokumentiert. Die für das Projekt relevanten Statusnummern sind im Folgenden kurz erläutert:

200(`OK`) Die Anfrage wurde erfolgreich abgearbeitet

201(`CREATED`) Ein Objekt wurde erfolgreich erstellt

400(`BAD REQUEST`) Wird zurückgegeben wenn eine Anfrage fehlerhaft ist. Gründe dafür könnte ein falsch codierter Body sein.

401(`UNAUTHORIZED`) Dieser Status wird zurückgegeben, wenn eine nicht autorisierte Anfrage getätigt wurde.

405(`METHOD NOT ALLOWED`) Wird zurückgegeben, wenn eine URL für das gesendete Verb nicht definiert ist.

409(`CONFLICT`) Dieser Status besagt, dass eine Anfrage nicht durchgeführt werden kann. Ein Beispiel dafür könnte sein, dass ein Unternehmen bereits existiert und deshalb nicht neu erstellt werden kann.

Die konkrete Umsetzung der Schnittstelle wird im Backend in dem Package `controller` umgesetzt. Darin wird für jede HTTP-Anfrage eine Methode definiert, welche bei jedem Request ausgeführt wird. Als Rückgabewert dieser Methoden wird eine Instanz der `javax.ws.rs.core.Respons` Klasse zurück gegeben. Diese enthält einen der oben stehenden Statusnummern sowie wie eine Entität, welche im Body der Serverantwort steht.

Dort befinden sich sechs verschiedene Klassen, welche die REST-Anfragen abfängt. Die **GameController**-Klasse kümmert sich dabei, um alle generellen Aufrufe, welche das Spiel und die Authentifizierung konfigurieren. Die **CompanyController**-Klasse kümmert sich um alle Anfragen an die URL `rest/companies`. Diese dient dazu unternehmensspezifische Informationen, anzufragen oder Unternehmen zu konfigurieren. Für die jeweiligen Abteilungen des Unternehmens gibt es weiter Controllerklassen.

3.2.1 Token - Autorisierung

Da es unsicher wäre jede Anfrage ohne Authentifizierung durchzuführen, gilt es bei vielen Funktionen zu überprüfen, ob eine bestimmte Anfrage sicher durchgeführt werden kann. Im Falle des Projektes wird als Authentifizierung der Name des Unternehmens, sowie ein Passwort verwendet. Will sich nun ein User im Namen eines Unternehmens anmelden, so muss dieser eine `POST`-Anfrage mit dem Unternehmensnamen und zugehörigen Passwort an die URL `/rest/` senden. Die Anfrage empfängt die `authenticateUser`-Methode der `GameController`-Klasse, welche die Zugangsdaten als Parameter übergeben bekommt. (siehe Quelltext 3.1) Anschließend werden die übergebenen Zugangsdaten überprüft. Dies wird mithilfe der `authenticate()`-Methode getan.(siehe (*@@*)) Sollten die Benutzerdaten nicht stimmen, so wirft die `authenticate`-Methode eine Exception und gibt eine Response mit dem Status 401(Unauthorized) zurück. Im Falle, dass die Daten valide sind, wird ein Token generiert und anschließend mit Status 200 zurückgegeben.

Auf das Generieren des Tokens wird aus Platzgründen nicht weiter eingegangen. Damit der Benutzer nur auf sensible Methoden zugreifen kann, muss dieser den generierten Token bei jeder Anfrage als Headerparameter mitgeben werden. Direkt vor dem Token muss das Wort `Bearer` stehen. Dieses gibt an, um welche Art der Verschlüsselung es sich handelt. Wenn eine Anfrage den Server erreicht, muss dieser gegebenenfalls den Headerparameter überprüfen. Dafür dient die `AuthentificationFilter`- Klasse. Sie ist eine Unterklasse eines `ContainerRequestFilter` und sorgt dafür, dass bei jedem Methodenaufruf der Headerparameter `Authorization` validiert wird. Zusätzlich dazu ermöglicht dieser Filter eine eindeutige Identifizierung des Unternehmens. Da ein Benutzer bloß sein eigenes Unternehmen

```
@POST
@Produces("application/json")
@Consumes("application/x-www-form-urlencoded")
public  Response authenticateUser(@FormParam("username") String companyName, @FormParam("password") String password) {
    try {
        authenticate(companyName, password);
        String token = issueToken(companyName);
        return Response.ok(token).build();
    } catch (Exception e) {
        return Response.status(Response.Status.UNAUTHORIZED).build();
    }
}
```

Quelltext 3.1: `authentivateUser()` Methode der `GameController`-Klasse

konfigurieren darf, wird aus diesem Grund der Token als Validierung herangezogen. Um eine Methode nun zu sichern, wird ein Dekorator namens `@Secured` angegeben. Jede Methode, vor welcher diesen Dekorator steht, führt zuvor den `AuthentificationFilter` aus. Als Parameter kann man nun eine Instanz eines SecurityContextes erhalten. Durch diesen ist es nun möglich das eine Instanz des Unternehmens zu erhalten zu dem der Token zugewiesen ist. (siehe Quelltext 3.2),

```

    @GET
    @Secured
    public Response getCompany(@Context SecurityContext securityContext) {
        return Response.status(200).entity(gson.toJson(getCompanyFromContext(securityContext))).build();
    }

```

Quelltext 3.2: Beispiel des `@Secured`-Dekorators

3.3 Angular JS Markus Böbel

Die Präsentationsschicht des Unternehmensplanspiels besteht wie in Kapitel 2.2 beschrieben aus einer einfachen Webanwendung, einer Single Page Application (SPA). Dies ist eine Webseite, welche grundlegend nur aus einer einzelnen Seite besteht. Erforderliche Daten werden zur Laufzeit nachgeladen und angezeigt. Somit entsteht eine eigenständige Webanwendung.

Für diese Dynamik wird des Öfteren die Webprogrammiersprache JavaScript verwand. Diese ist eine asynchrone Programmiersprache. Der Unterschied zu einer synchronen Sprache zeigt sich im Beispiel einer simplen HTTP-Abfrage. Während synchrone Sprachen einen Aufruf starten und bis zum Erhalten der Antwort warten, fahren asynchrone Sprachen fort und realisieren das entgegennehmen durch `Callbacks`, also Funktionen, welche nachträglich aufgerufen werden, und die Daten übergeben bekommen. Der Vorteil darin besteht, dass die Anwendung flüssiger läuft, da der Programmablauf zu keinem Zeitpunkt gesperrt werden kann. Die schnell wachsende Komplexität des Quellcodes relativiert die genannten Punkte. Im Umkehrschluss würde die Implementierung einer solchen Single Page Application sehr komplex und unübersichtlich werden.

Um dagegen vorzugehen, wurden bestimmte Frameworks entwickelt die Entwickler Möglichkeiten geben, solche Webanwendungen strukturiert und übersichtlich zu entwerfen und implementieren. Eines solcher Frameworks ist Googles AngularJS, welches speziell für die Entwicklung von SPAs entwickelt wurde. Eine Besonderheit von Angular 2 ist es, dass anstatt reinem JavaScript die Anwendung in TypeScript geschrieben wird. TypeScript ist eine Erweiterung des ECMA-Script 6 JavaScript Standards und ermöglicht zum einen das Typisieren von Variablen, sowie das Beschreiben von Objekten mit Hilfe von Dekoratoren. (Erkennbar durch ein `@`- Zeichen) Bevor die Webanwendung gestartet werden kann, gilt es das TypeScript wieder in natives JavaScript zu kompilieren.

3.3.1 Aufbau einer Angular 2 Anwendung

Während die erste Version von Angular Webanwendungen durch vereinfachte JavaScript-Skripte entwickelt wurden, ist Angular 2 nun Komponenten-basiert. Dies bedeutet, dass die komplette Anwendung in verschiedene Komponenten unterteilt wird, welche eigene Funktionalitäten besitzen. Werden in einem Teil der Webseite zum Beispiel Informationen über ein Unternehmen angezeigt, so wird dies in einem Komponenten beschrieben. In anderen Worten besteht die komplette Anwendung aus ineinander-verschachtelten Komponenten. Neben Komponenten gibt es noch weitere Elemente:

Services Services dienen dazu Daten anzufragen. Angular 2 arbeitet nach dem Model-View-Controller (MVC)-Pattern, die Services dienen dabei als Model. Sie fragen Daten an und stellen sie den einzelnen Komponenten bereit. Somit werden alle Anfragen an die REST Schnittstelle in Services getätigt. In anderen Worten entsteht eine Verbindung zum Backend nur durch Services.

Components Wie oben beschrieben dienen Komponenten dazu die Applikation in viele kleine Teile zu teilen. Sie besitzen ein HTML Template und einen `Selector`. Der `Selector` dient dazu den Komponenten anzuzeigen. Dies geschieht über die einfache Erwähnung in einer anderen HTML Seite, beziehungsweise in dem Template eines weiteren Komponenten. Der `Selector 'test-component'` kann somit in anderen HTML Files mit dem Tag `<test-component></test-component>` eingebunden werden. Zwischen den Tags ist es möglich eine Ladeanzeige zu implementieren. Diese wird solange angezeigt, wie der Komponent geladen wird.

Modules Ein Modul ist das oberste Element. In ihm werden alle Elemente wie Komponenten oder Services definiert und integriert. Nur wenn ein Element im entsprechenden Angular-Modul initialisiert wird, kann es verwendet werden. Jedes Modul enthält eine Bootstrapkomponente, welche bei Start des Moduls angezeigt wird.

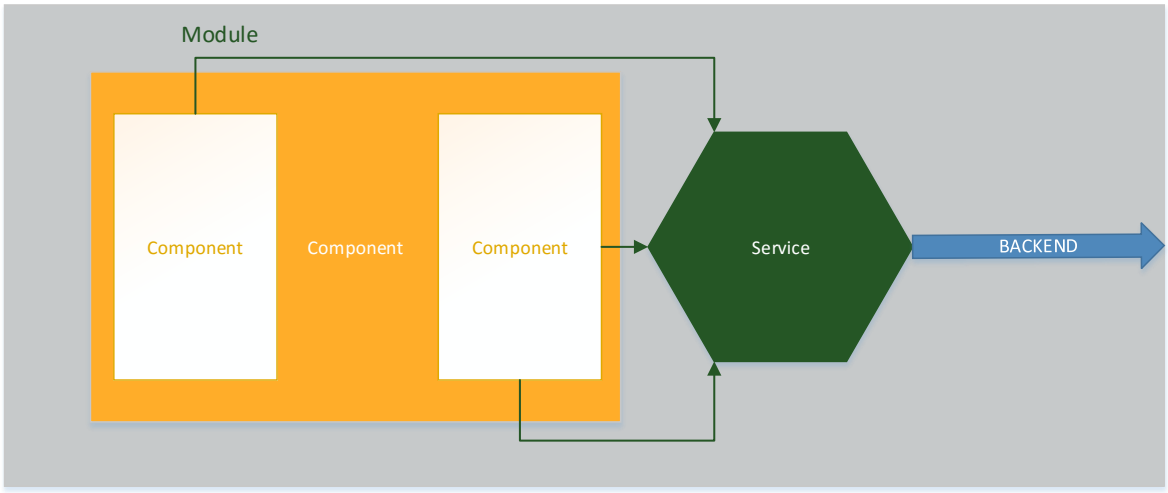

Abbildung 3.1: Aufbau einer Angular 2 Applikation

Im Falle des Fallspieles gibt es 2 Module, welche nebeneinander agieren. Zum einen das `App`-Modul. Dieses beinhaltet alle Elemente, welche sich um das Login Fenster kümmern. Dazu gehören die entsprechenden Komponenten zur Darstellung der Highscore- oder Spielerliste. Wenn sich der Spieler erfolgreich einloggt gelangt er auf die letztendliche Spielseite. Diese wird strukturiert von `Home`-Modul. In diesem befinden sich alle Komponenten für die einzelnen Unterseiten.

3.3.2 Komponente der Produktion

Aufgrund der begrenzten Länge der Arbeit wird im Folgenden nur die Komponente der Produktion tiefer erläutert. Diese befindet sich im `Home-` Modul und wird innerhalb der Home-Komponente angezeigt. Der Quelltext 3.3 zeigt den Grundaufbau einer Komponente. Zuerst wird im obligatorischen `@Component`-Dekorator der Selector (`home-component`) angegeben. Ein Zusammenschluss verschiedener Wörter wird dabei über Bindestriche geregelt. Direkt gefolgt davon wird das Template der Komponente angegeben. Dies kann entweder direkt HTML Code sein, oder der Verweis auf eine seperate HTML Datei. Die letztendliche Komponente wird im Beispiel durch die Klasse `ProduktionComponent` widergespiegelt. Das Schlüsselwort `export` ist dabei ein äquivalent zum `public` in Java. Im Konstruktor der Klasse werden Services übergeben. Diese müssen vorab entweder als Provider im `@Component`-Dekorator oder im entsprechenden Modul der Komponente angegeben werden. Im Beispiel der Produktionskomponente `ProduktionComponent` wird der `ProduktionService`, welche für die Abteilung der Produktion zuständig ist, sowie der `HRService`, welcher zum Beispiel für das Einstellen oder Feuern von Personal dient, definiert. Durch das Schlüsselwort `private` werden die beiden Services als Attribut der Klasse hinterlegt und können von nun an überall in der Komponente verwendet werden.

```
1  @Component({
2      selector    : 'home-component',
3      templateUrl: '../../../../templates/components/home.
   components/produktion.component.html',
4  })
5
6  export class ProduktionComponent {
7      employees;
8      errorMaschinen;
9      errorLinie;
10     sold;
11
12     constructor(private proService:ProduktionService ,private
   hrService:HRService)
13     {
14         [...]
15     }
16     [...]
17 }
```

Quelltext 3.3: Grundaufbau einer Komponente

Die Seite der Abteilung Produktion besteht aus mehreren Abschnitten. So soll der Benutzer

Lagerhäuser, Produktionsstätten kaufen, sowie seine Maschinen erwerben, verwalten, reparieren und verkaufen können. Des Weiteren soll der Benutzer die Möglichkeit bekommen neue Produktlinien in Produktion zu geben. Diese Funktionen können nun in weitere kleinere Unterkomponenten aufgeteilt werden. So entsteht nun innerhalb der Produktionskomponente folgende neue Unterkomponenten: (siehe Abbildung 3.2)

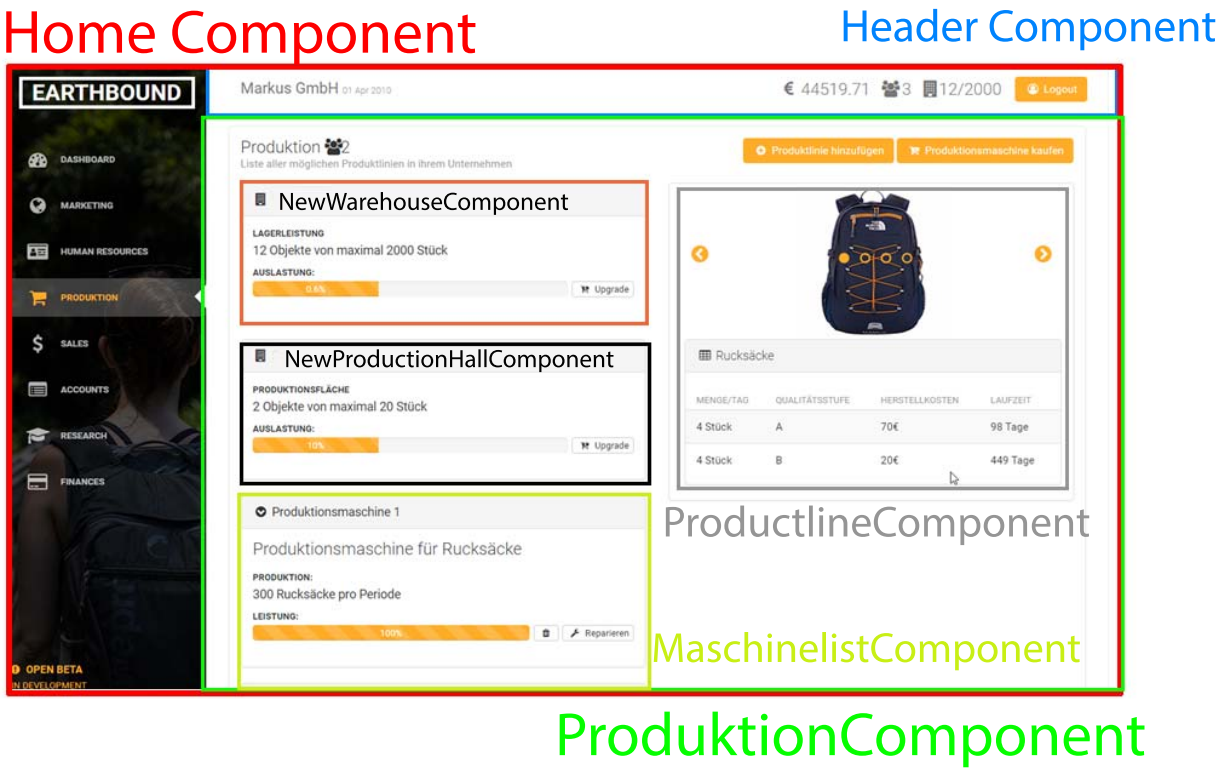

Abbildung 3.2: Einteilung der Produktionskomponente

NewWarehouseComponent Diese Komponente dient zum Kauf von Lagern und zur Ansicht der Lagerauslastung.

NewProduktionHallComponent Diese Komponente ähnelt der des Lagerhauses, dient dennoch zur Verwaltung von Produktionsstätten.

MachineListComponent Die Liste der Maschinen wird in dieser Komponente visualisiert. Zudem hat der Spieler zusätzlich die Möglichkeit seine Maschinen zu reparieren oder zu verkaufen. Zudem sieht der Spieler die Abnutzung seiner Maschinen.

ProductLineListComponent Diese Komponente dient lediglich zur Visualisierung der aktuellen Produktionen. Dabei hat der Spieler die Möglichkeit zwischen den vier verschiedenen potentiellen Produktionsprodukten hin- und herzuschalten.

NewProductLineComponent Auf den ersten Blick versteckt ist die Komponente zur Erstellung neuer Produktlinien. Diese ist ein genanntes Modal, also ein Fenster, welches sich öffnet sofern der Spieler auf den Produktlinie hinzufügen Button auf der oberen rechten Seite der Anwendung klickt.

NewMachineComponent Direkt neben dem Button zur Erstellung neuer Produktlinien, befindet sich ein Knopf zum Erwerb neuer Produktionsmaschinen. Dieser öffnet ebenfalls ein solches Modal.

Die oben beschriebenen Komponenten besitzen einen identischen Grundaufbau verglichen mit dem der Produktionskomponente. Über den angegebenen `selector` können die Komponenten nun wie oben bereits erwähnt in die Produktionskomponente geladen werden. Die einzelnen Komponente agieren dabei wie ganz normale HTML - Blocktags. (siehe Quelltext 3.4)

```
[...]
<div class="content">
   <div class="row">
      <div class="col-md-6">
         <new-warehouse-component>
            Loading
         </new-warehouse-component>
         <new-production-hall-component>
            Loading
         </new-production-hall-component>
         <machine-list-component>
            Loading
         </machine-list-component>
      </div>
      <div class="col-md-6">
         <div class="panel panel-default">
            <productline-list-component>
               Loading
            </productline-list-component>
         </div>
      </div>
   </div>
</div>
[...]
```

Quelltext 3.4: Ausschnitt des Templates der Produktionskomponente

Wenn nun Daten in die auf dem Interface angezeigt werden, welche als Variable in der Komponente bestehen, dann geht das über die Verwendung geschweifter Klammern. Soll nun

zum Beispiel die Variable `x` ausgeben werden ist dies über `x` möglich. Neben der einfachen Ausgaben unterstützt Angular 2 auch Kontrollstrukturen wie Schleifen (`*ngFor`) oder Verzweigungen(`*ngIf`). Als Beispiel zeigt die Abbildung 3.5 einen Auszug der Komponente für das Anzeigen und Verwalten von Maschinen.

```
1  <div *ngFor="let machine of data; let i = index" class="panel-
   group" id="accordion" role="tablist" aria-multiselectable="true">
2      [...]
3            <p *ngIf="machine.produkt == 'Duffel'">{{machine.
   kapazitaet}} Duffels pro Periode</p>
4       [...]
5  </div>
```

Quelltext 3.5: Beispiel für das Anzeigen von Daten in Angular

Um Daten anzeigen zulassen, müssen diese aus dem Backend geladen werden. Die eigentliche Anfrage findet im Service statt. Dieser stellt nun eine Methode zur Verfügung, mithilfe dessen man die Daten anfragen kann. Als Rückgabewert wird ein sogenanntes `Observable` zurück gegeben. Nun kann man dieses `Observable` abonnieren, das heißt wenn die Anfrage vom Backend beantwortet wird, führt das `Observable` eine bestimmte Funktion aus. Der Quelltext 3.6 der Maschinenlistenkomponente zeigt das Füllen der in Quelltext 3.5 verwandten Variable `data`.

```
1  export class MachineListComponent {
2      data;
3
4      constructor(private _proService: ProduktionService,[...]) {
5          this.loadList();
6          [...]
7      }
8
9      loadList()
10     {
11         this._proService.getMachines().subscribe(data=>this.data =
   data);
12     }
13     [...]
14 }
```

Quelltext 3.6: Anfragen von Daten

So wird wie im Konstruktor des Komponenten zuerst der Produktionsservice als Attribut der Komponente definiert. Instanzen von Services fangen oftmals mit einem Unterstrich an. Der Konstruktor welcher zu Beginn der Lebenszeit der Komponente aufgerufen wird, führt nun

die Methode `loadList()` aus. Hier wird die Funktion getMachines() des Produktionsservices aufgerufen. Wie zuvor beschrieben gibt diese eine Instanz eines `Observabels` zurück. Dieses wird nun mit Hilfe der Methode `subscribe()` abonniert. Als Parameter besteht dabei die Möglichkeit drei Callbackfunktionen anzugeben. Die erste wird ausgeführt, sofern die Anfrage korrekt verlief, die zweite Callbackfunktion dient dazu potentielle Fehler abzufragen. Und als dritten Parameter kann eine Callbackfunktion angegeben werden, welche ausgeführt wird, sofern die Anfrage durchgeführt wurde. Im Falle des oberen Beispiels, wird bloß eine Funktion angegeben. Diese hat einen Parameter, welche den Rückgabewert der Anfrage enthält, der anschließend in die Variable `data` der Komponente geschrieben wird.

Die anderen Komponenten besitzen einen ähnlichen Aufbau und werden aus diesem Grund nicht weiter erläutert.

4 Tests

4.1 Ziel des Testens

Durch das Testen wird die Software dahingehend überprüft, ob zum einen Funktionalitäten einwandfrei funktionieren, aber auch ob die Handhabung mit der Software nutzerfreundlich ist. Zu diesen Zwecken wurde das Coding selbst überprüft, die Interfaces der Software, welche von den Spielern gesehen und verwendet werden und schließlich mithilfe eines Nutzerakzeptanztests die Usability.

4.2 Testen mit JUnit

Hier wurde mittels eines Modultests der Quellcode überprüft, indem einzelne Komponenten (engl. Units) auf das korrekte Verhalten hin kontrolliert wurden. Modultests gehören in der Software Entwicklung zu den sogenannten White-Box-Tests. Bei dem weiteren Vorgehen ist zu beachten, dass die Besonderheiten der Modultests darin liegen, dass sie unabhängig voneinander ausgeführt werden können. Dafür werden jegliche zu prüfenden Klassen in eigenständige Testeinheiten aufgenommen. Diese kann man dann entweder für sich alleine ausführen, oder aber zusammen mit allen anderen bestehenden Testeinheiten.

Hierfür wurde in unserem Projekt das JUnit Framework verwendet, welches bereits mit vielen bestehenden Funktionalitäten ausgestattet ist, um das Testen von Software, entwickelt in Java, zu optimieren. So gibt es die Möglichkeit, zum einen Ergebnisse von Tests darzustellen, aber auch den Deckungsgrad der Codezeilen, Methoden und Klassen, welche getestet wurden, abzubilden. Das Vorgehen bei dem Testen ist abhängig davon, welche Funktionalitäten innerhalb der Methoden abgebildet sind. Meistens wurden bestimmte Objekte nach ihren Eigenschaften hin untersucht, um festzustellen, ob nach Ausführung einer Methode die erwartete Veränderung stattgefunden hat. Das Projekt besitzt eine insgesamte Code-Abdeckung von : 60 %.

4.3 User Akzeptanztest

Abgesehen von dem Coding selbst wurden auch die Oberflächen getestet. Dies hat den Hintergrund, dass man wissen möchte wie die Software von Nutzern verwendet wird, die nicht an der Entwicklung beteiligt waren und welche in Zukunft mit dem Programm in Kontakt kommen werden. Diese Personen sollten dementsprechend keinerlei Vorkenntnis darüber haben, wie sie sich in den Oberflächen zu orientieren haben und wie man vorzugehen hat, wenn man eine Aktionskette ausführen möchte. Zur Überprüfung der Nutzbarkeit des Unternehmensplanspiels wurden drei Szenarien entwickelt, welche von einer Testperson ausgeführt werden sollten. Hierfür hat man die Aktionen verwendet, die typischerweise beim spielen mehr oder weniger häufig anfallen.

4.3.1 Erstes Szenario

In dem ersten Szenario soll sich der Nutzer auf eine Ausschreibung bewerben und danach bereits veranlassen, das Produkt für die Ausschreibung herzustellen. Dafür ist es nötig, zuerst einen HR Mitarbeiter einzustellen, damit man Angestellte in den Abteilungen Vertrieb und Produktion hinzufügen kann. Ist dies erledigt erhält, man die Möglichkeit, sich für eine Ausschreibung zu bewerben und auch zu produzieren, falls vorher eine Lagerhalle, eine Produktionshalle und eine Maschine mit den nötigen Anforderungen angeschafft worden ist. Um zu produzieren wird zuerst eine entsprechende Maschine des zutreffenden Types benötigt und eine Produktlinie mit der erforderlichen produktqualität, der Laufzeit und der Ausbringungsmenge.

Ergebnis Der Proband hat sich schnell auf der Startseite orientiert und direkt ein Unternehmen erstellt. Die Buttons um Mitarbeiter hinzuzufügen konnten schnell ausgemacht werden und nachdem alle benötigten Mitarbeiter hinzugefügt wurden, ist der Nutzer zielstrebig zu der Abteilung Sales übergegangen. Hier hat er sich für das erste Ausschrieben entschieden und sich unmittelbar beworben. Als es dann zur Produktionsklasse weiterging, wollte der Nutzer zuerst eine Maschine, ohne vorher für entsprechenden Platz gesorgt zu haben, kaufen. Als dies nicht funktionierte, sah er sich das Interface für wenige Sekunden an und wählte dann eigenständig die Produktions- und Lagerhalle aus. Trotz der Erklärung wollte der Nutzer zuerst eine Produktlinie starten, durch die Fehlermeldung wusste er aber sofort, dass erst eine Maschine benötigt wird.

Vorgeschlagene Veränderungen Fehlermeldungen noch spezifischer machen. Sie sollen den Nutzer anleiten, was genau gemacht werden muss. Auch könnten Buttons ausgegraut werden, falls die Vorbedingungen noch nicht erfüllt sind (Beispiel: erst Maschine und dann Produktionslinie).

4.3.2 Zweites Szenario

In dem zweiten Szenario soll der Nutzer für das produzierte Produkt ein Forschungsprojekt starten, um dessen Herstellungskosten zu senken. Hierfür wird zuerst ein Angesteller in der Abteilung Forschung benötigt, welchen man dann dem zu startenden Projekt hinzufügen kann. Desweiteren soll ein Kredit in Höhe von 15000 für 10 Monate aufgenommen werden. Dafür ist es nötig zuerst einen Angestelten in Finanzen hinzuzufügen.

Ergebnis Der Proband hatte hier keinerlei Probleme, die Anforderungen umzusetzen. Die Schritte wurden schnell und größtenteils sicher ausgeführt. Nur zur Kreditaufnahme wurde der Button erst drei bis vier Sekunden später entdeckt, da er sich relativ weit unten am Rand befindet.

Vorgeschlagene Veränderungen Vorgeschlagene Veränderungen: Verrücken des Buttons auf der Finanzen Seite, nach oben oder den rechten Rand.

4.3.3 Drittes Szenario

In dem letzten Szenario wird der Nutzer schließlich aufgefordert, ein Weihnachtsgeld für die Mitarbeiter einzuführen und eine Social Media Marketingkampagne zu starten. Hierfür müssen wieder vorerst die benötigten Mitarbeier den entsprechenden Abteilungen hinzugefügt werden.

Ergebnis Die Marketingkampagne konnte schnell und ohne Probleme gestartet werden. Allerdings wusste der Nutzer nicht wie er vorzugehen hat, um soziale Leistungen freizuschalten. Nachdem er wenige Sekunden überlegt hat, wählte er die Abteilung Human Recources aus, mit der Begründung, dass die anderen Abteilungen aufgrund ihrer Bezeichnungen unpassend erschienen.

Vorgeschlagene Veränderungen Die fehlenden Angaben aus der Szenario-Beschreibung können noch bei weiteren Testverfahren ergänzt werden. Ansonsten gab es hier keine Vorschläge zu dem Produkt.

4.3.4 Nutzerbeschreibung und Feedback

Bei der Testperson handelt es sich um einen Mann Mitte 20 Jahre, welcher selbst beruflich in der Wirtschaft tätig ist und hobbymäßig Videospiele spielt. Das Feedback zum Spiel war positiv, und besonders gefiel ihm die Möglichkeit, sich erst auf Ausschreibungen bewerben zu müssen. Das Layout der Navigationsbar fand er übersichtlich, genauso wie die Aufteilung der anderen Oberflächen. Die Farbe fand er zwar etwas markant, empfand dies aber nicht als negativ. Auch die Anzeige des Dashboard wurde sehr positiv kommentiert, wie sie die Kennzahlen des Unternehmens dynamisch abbildet.

A Testanhang

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean

placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

A.1 Subtestanhang

B Noch ein Testanhang

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich:

- dass ich die vorliegende Arbeit mit dem Thema *Titel der Arbeit* selbständig verfasst und
- keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
- Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Ort, Datum Unterschrift