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

Kurzfassung

Titel: Der Titel der Arbeit
Verfasser: Erika Musterfrau

Hier können Sie eine Kurzfassung der Arbeit verfassen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

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 3.1). Diese wird auf einem Tomcat Server initialisiert, auf welchem die Anfragen bearbeitet und das Spielkonzept umgesetzt werden.

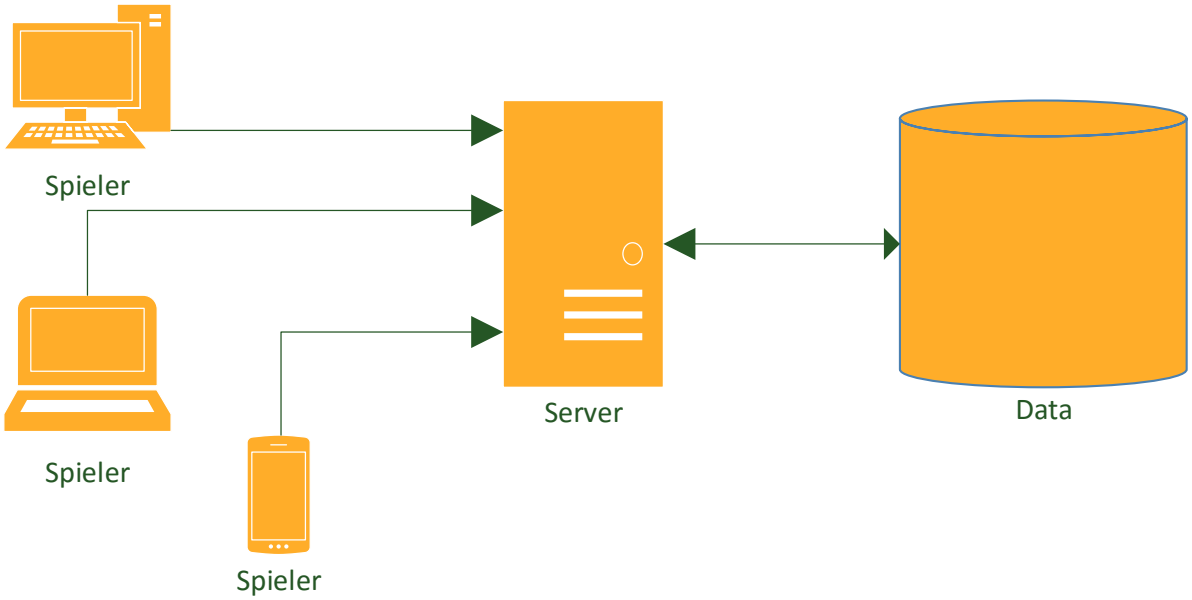

Abbildung 2.1: Aufbau der Applikationsarchitektur

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 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.

3.1.1 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.

```
1  @POST
2  @Produces("application/json")
3  @Consumes("application/x-www-form-urlencoded")
4  public  Response authenticateUser(@FormParam("username") String
       companyName, @FormParam("password") String password) {
5      try {
6          authenticate(companyName, password);
7          String token = issueToken(companyName);
8          return Response.ok(token).build();
9      } catch (Exception e) {
10         return Response.status(Response.Status.UNAUTHORIZED).build();
11     }
12 }
```

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

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 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),

```
1
2     @GET
3     @Secured
4     public Response getCompany(@Context SecurityContext
    securityContext) {
5        return Response.status(200).entity(gson.toJson(
    getCompanyFromContext(securityContext))).build();
6     }
7
```

Quelltext 3.2: Beispiel des `@Secured`-Dekorators

3.2 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)

3.2.1 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.

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.

4 Tests

4.1 Ziel des Testens

blablabla

4.2 Whitebox Testing

blublublu

4.2.1 JUnit

jujuju

4.3 Akzeptanztest

akakakak

4.4 Forschung

jojojo

4.4.1 Forschungsprojekt

jajaja

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