Zero1one

Dokumentation von Gruppe 2

vorgelegt am 28. Januar 2020

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs [Kursbezeichnung]

von

[Vorname Nachname]

|  |  |
| --- | --- |
| Betreuer in der Ausbildungsstätte: | DHBW Stuttgart: |
| [Name des Unternehmens] [Titel, Vorname und Nachname d. Betreuers] [Funktion des Betreuers] [Unterschrift des Betreuers] | [Titel, Vorname und Nachname des wissenschaftl. Betreuers/Prüfers] |

**Inhaltsverzeichnis**

[Abkürzungsverzeichnis (bei Bedarf) III](#_Toc31093439)

[Abbildungsverzeichnis (bei Bedarf) IV](#_Toc31093440)

[Tabellenverzeichnis (bei Bedarf) V](#_Toc31093441)

[1 Einleitung 1](#_Toc31093442)

[2 Hauptteil 2](#_Toc31093443)

[2.1 Eine Abschnittsüberschrift 2](#_Toc31093444)

[2.2 Zweiter Abschnitt 2](#_Toc31093445)

[2.2.1 Ein Abbildungsbeispiel 2](#_Toc31093446)

[2.2.2 Ein Tabellenbeispiel 3](#_Toc31093447)

[3 Nächstes Kapitel 4](#_Toc31093448)

[3.1 Abschnitt 4](#_Toc31093449)

[3.1.1 Unterabschnitt 4](#_Toc31093450)

[3.1.2 Unterabschnitt 4](#_Toc31093451)

[3.2 Entsprechend 4](#_Toc31093452)

[3.2.1 Fortgesetzt 4](#_Toc31093453)

[3.2.2 Nochmal fortgesetzt 4](#_Toc31093454)

[4 Schlussteil 5](#_Toc31093455)

[Literaturverzeichnis 8](#_Toc31093456)

# Abkürzungsverzeichnis (bei Bedarf)

AktG = Aktiengesetz

BFH = Bundesfinanzhof

# Abbildungsverzeichnis (bei Bedarf)

[Abb. 1: Eine Abbildung 2](#_Toc31092940)

# Tabellenverzeichnis (bei Bedarf)

[Tab. 1: Eine Beispieltabelle 3](#_Toc31092943)

1. Einleitung
2. Thymleaf

Dimis Teil

* 1. Einsatz von Thymleaf im Programm

Thymleaf wurde bei uns auf nahezu allen Seiten des Programms eingesetzt, da wir fast ausschließlich dynamische Inhalte haben. Unter dynamischen Inhalten oder dynamischen Webseiten versteht man Webseiten, dessen Inhalt sich häufig ändert. Aus diesem Grund ist es notwendig, die Webseite erst nach Aufruf durch den Nutzer entsprechend aufzubauen.

(Quelle: <https://books.google.de/books?hl=de&lr=&id=VlCMzDrSTyYC&oi=fnd&pg=PA5&dq=dynamischer+webseiten&ots=V-nqQMktM6&sig=LV5yU6wqLHejZij5G_6W6AFIrbw&redir_esc=y#v=onepage&q=dynamischer%20webseiten&f=false> Seite 23)

Lediglich auf Seiten mit statischem Inhalt, wie z.B. die Hilfe, war der Einsatz nicht nötig. Generell haben wurde sich darauf verständigt, HTML als gemeinsame Grundlage zu verwenden. Wie bereits im Kapitel zu HTML und CSS beschrieben, wurden die HTML-Dateien von den Kollegen aus dem Front-End entwickelt und uns zur Verfügung gestellt. Wir haben diese dann in die Ordnerstruktur von Thymleaf übernommen und entsprechend für die Darstellung der Inhalte angepasst.

Die HTML-Dateien wurden im Java-Projekt im Ordner „src/main/resources/templates“ als so genannte „Templates“ hinterlegt. Zusätzliche Dateien wie z.B. Bilder oder Skripte (vorwiegend CSS-Dateien) die bereits von den Kollegen in die HTML-Dateien eingebunden wurden, sind unter dem selben Pfad, allerdings nicht im Ordner „templates“ sondern im Ordner „static“, abgelegt. Der Ordner „static“ beinhaltet somit alle Dateien, die nicht von der Java-Seite angefasst werden. Diese Inhalte werden beim Aufruf der Seite automatisch über das HTML entsprechend eingebunden.

Die HTML-Dateien, die in diesem Abschnitt als „Templates“ bezeichnet werden, werden durch einen Controller aufgerufen. Ein Controller in einer Thymleaf- bzw. Spring-Web-Umgebung, ist eine Java-Klasse, die auf ein HTTP-Request reagiert und dem HTTP-Response ein entsprechendes HTML-Template sowie die gewünschten Inhalte übergibt. Das nennt man auch Mapping. Diese Controller-Klasse ist also direkt an die Webschnittstelle angebunden. (Siehe hierzu Kapitel X.X Schnittstelle).

In unserem Projekt befindet sich die Controller-Klasse in dem Java-Package „com.gruppezwei.zero1one.controller“. Zur besseren Übersicht der einzelnen Mappings, haben wir diese in separate Controller-Klassen ausgelagert. Es gibt einen „HomeController“ für das Dashboard, einen „anlegenController“ für alle Seiten die Funktionen zum Anlegen und Speichern beinhalten, sowie einen „userController“ der alle Mappings für Funktionen zu Benutzereigenschaften und für statische Seiten beinhaltet.

1. Schnittstelle

Wie bereits aus den vorherigen Kapiteln hervorgeht, wird das Java-Programm über eine webbasierte Oberfläche bedient. Das hat den Vorteil, dass auch ein „Server-Client-Betrieb“ möglich wäre. Um die Daten zwischen dem Backend (Java-Applikation) und dem Frontend (Webbasiertes User Interface) zu transferieren, verfügt die Software über eine REST-Schnittstelle. REST steht für Representational State Transfer und definiert eine Schnittstelle für Server-Client Anwendungen. Für den Austausch von Informationen wird das HTTP-Protokoll (Hyper-Text-Transfer-Protocol) eingesetzt. Der Zugriff auf die Schnittstelle erfolgt über so genannte URIs oder auch URLs, wie man es auch bei Webseiten des Internets kennt. Die REST-Schnittstelle wird durch folgende Kriterien definiert:

1. Client-Server

Es erfordert die Abtrennung von Client und Server und somit auch die Aufsplittung in Front- und Backend. Dabei stellt das Backend die Funktion und die Daten zur Verfügung, die vom Frontend genutzt wird.

1. Zustandslosigkeit

Zustandslosigkeit ist gegeben, wenn keine Daten des Clients auf dem Server gespeichert werden. Das setzt allerdings voraus, dass bei jeder Anfrage des Clients alle Daten im Request mitgesendet werden müssen. Das erhöht den Datenverkehr im Netzwerk, hat aber gleichzeitig den Vorteil, dass sich Fehler auf einen einzelnen Request beziehen und nicht von vorherigen Daten beeinflusst werden.

1. Caching

Um das Netzwerk zu entlasten, sollen alle Daten, die zwischengespeichert werden können, entsprechend gekennzeichnet werden. Dadurch kann der Client diese Daten „cachen“ und der Server muss ggf. nicht alle Daten bei einem erneuten Request übermitteln.

1. Einheitliche Schnittstelle

Die REST.-Schnittstelle ist eine einheitliche Schnittstelle, die sicherstellt, das ein bestimmter Request immer das das gleiche Ergebnis liefert. Außerdem ist die Implementierung von den restlichen Teilen der Software entkoppelt, was eine parallele Entwickelung der Software ermöglicht.

1. Mehrschichtige Systeme

Die Mehrschichtigkeit der Software soll die Skalierbarkeit und die Vereinfachung der Architektur der Software ermöglichen. Dabei sollen alle Komponenten der Software unabhängig voneinander funktionieren. Das ermöglicht auch eine leichte und agile Anpassung der Software.

1. Code on Demand

Dieses Prinzip ist optional und ermöglicht dem Client das Herunterladen des Codes. Der Code kann durch den Client selbst angepasst werden, ohne dass der Server alle möglichen Fälle abdecken muss.

Für die Kommunikation über das HTTP-Protokoll stehen folgende CRUD-Methoden (Create, Read, Update, Delete) zur Verfügung:

* Get 🡪 Fordert Daten vom Server an
* Post 🡪 Übermittelt Daten an den Server
* Put 🡪 Ändert bestehende Daten auf dem Server
* Delete 🡪 Löscht Daten auf dem Server

Außerdem bietet HTTP auch Statuscodes, wie z.B.:

* 100 Informationen
* 200 Erfolgreiche Operation
* 300 Umleitung
* 400 Client Fehler
* 500 Server Fehler

Diese Codes werden durch weitere Ziffern spezifiziert. So steht 404 z.B. für „Not found“ oder 403 für „Forbidden“. Ersteres signalisiert, dass die angeforderte Ressource nicht gefunden wurde, letzteres, dass der Client nicht über ausreichend Rechte verfügt.

Außerdem beinhaltet eine http-Nachricht einen Header und einen Body. Der Header übermittelt alle möglichen Meta-informationen, der Body die eigentlich abgefragten Informationen. Dabei wird der Request-Header nur bei Anfragen an den Server und der Response-header nur bei Antworten von dem Server mitgesendet. Auf die Unterschiede der beiden Header soll in dieser Arbeit aber nicht weiter eingegangen werden, da diese Methodik in der Software nicht verwendet wurde.

(Quelle: <https://tschutschu.de/resources/tschutschu/docs/SS2019/SS2019_T07_Wolf_Manuel_Studienarbeit.pdf> Seite 4 -7)

In unserem Projekt wurden generell die REST-Prinzipien eingehalten, wobei einzelne Prinzipien noch weiter vertieft und optimiert werden können. Für die Implementierung der Schnittstelle haben wir ein eigenes Controller-Package angelegt, wie es in Kapitel X.X Thymleaf bereits beschrieben ist. Eine entsprechende Dependency musste nicht ergänzt werden. Zum Ausführen des Programmes während der Entwicklung, stellt Spring einen Apache Tomcat Server zur Verfügung. Dieser simuliert den Server, auf dem die Anwendung läuft. Über den Webbrowser kann dann lokal mit der Software interagiert werden.