Zero1one

Dokumentation von Gruppe 2

vorgelegt am 23. Oktober 2020

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs WWI2019D

von

Daniel Reimann, Dimitri Wagner, Gianna Dihpol, Johanna Gröll, Laura Biberle,

Marcel Mahlfeld, Max Schneider, Pascal Jung, Til Wohlmuth

|  |
| --- |
| DHBW Stuttgart: |
| Marc Mössinger |

**Inhaltsverzeichnis**

[1 Einleitung 1](#_Toc54285783)

[2 Aufgabenstellung des Auftraggebers 2](#_Toc54285784)

[2.1 Anforderungsanalyse 2](#_Toc54285785)

[2.2 Pflichtenheft 3](#_Toc54285786)

[2.3 Modelle 4](#_Toc54285787)

[2.3.1 Use-case-Diagramm 4](#_Toc54285788)

[2.3.2 Klassen-Diagramm 7](#_Toc54285789)

[2.3.3 Entity-Relationship-Modelle 9](#_Toc54285790)

[3 Organisation 11](#_Toc54285791)

[3.1 Projektorganisation 11](#_Toc54285792)

[3.2 Git 12](#_Toc54285793)

[3.3 Projektphasen 13](#_Toc54285794)

[3.3.1 Die Organisations- und Programmentwurfsphase 13](#_Toc54285795)

[3.3.2 Die Erstellungsphase 13](#_Toc54285796)

[3.3.3 Die Zusammenführungs- und Anpassungsphase 14](#_Toc54285797)

[3.3.4 Die Test- und Dokumentationsphase 14](#_Toc54285798)

[3.3.5 Die Fertigstellungsphase 14](#_Toc54285799)

[3.4 Agiles Projektmanagement 14](#_Toc54285800)

[3.4.1 Scrum 15](#_Toc54285801)

[3.4.2 Vorgehensweise 15](#_Toc54285802)

[3.4.3 Das Scrum-Team 17](#_Toc54285803)

[4 Programmiersprachen und Erweiterungen 19](#_Toc54285804)

[4.1 HTML 19](#_Toc54285805)

[4.2 CSS 20](#_Toc54285806)

[4.3 JavaScript 21](#_Toc54285807)

[4.4 Java 22](#_Toc54285808)

[4.5 Thymeleaf 23](#_Toc54285809)

[4.6 Spring 24](#_Toc54285810)

[4.6.1 Gründe für die Nutzung von Spring 25](#_Toc54285811)

[4.6.2 Erweiterungen von Spring 26](#_Toc54285812)

[5 Datenbanken 29](#_Toc54285813)

[5.1 Auswahl des Datenbankmodells 29](#_Toc54285819)

[5.1.1 Hierarchische Datenbank 30](#_Toc54285820)

[5.1.2 Vernetzte Datenbank 30](#_Toc54285821)

[5.1.3 Objektorientierte Datenbank 31](#_Toc54285822)

[5.1.4 Relationale Datenbank 32](#_Toc54285824)

[5.1.5 Aufbau der Datenbank 33](#_Toc54285825)

[5.1.6 Auswahl des Datenbankmanagementsystems 35](#_Toc54285826)

[5.1.7 Einbau der Datenbank in das Projekt 36](#_Toc54285827)

[6 Schnittstellen 37](#_Toc54285828)

[7 Beschreibung des Quellcodes 39](#_Toc54285829)

[7.1 Frontend 39](#_Toc54285830)

[7.1.1 Entwicklungsablauf 39](#_Toc54285831)

[7.1.2 Beschreibung der technischen Umsetzung 41](#_Toc54285832)

[7.1.3 Quellcode der Unterseiten 45](#_Toc54285833)

[7.2 Backend / Webanbindung 47](#_Toc54285834)

[7.2.1 Auslieferung 47](#_Toc54285835)

[7.2.2 Zentrales Management 49](#_Toc54285836)

[7.2.3 Datenbankzugriff 50](#_Toc54285837)

[7.2.4 Dateien, Struktur und Übermittlung 53](#_Toc54285838)

[7.2.5 Darstellung auf dem Client 54](#_Toc54285839)

[7.2.6 Sicherheit 55](#_Toc54285840)

[7.2.7 Fehlerhandling 56](#_Toc54285841)

[7.3 Codequalität 57](#_Toc54285842)

[7.4 Qualitätssicherung 58](#_Toc54285843)

[8 Berechtigungsverwaltung 62](#_Toc54285844)

[8.1 Berechtigungsverwaltung generell 62](#_Toc54285845)

[8.2 Berechtigungsverwaltung im Programm 62](#_Toc54285846)

[9 Benutzerhandbuch 64](#_Toc54285847)

[Literaturverzeichnis 66](#_Toc54285848)

1. Einleitung

Durch die Information Technology Infrastructure Library (ITIL), welche als eines der meist genutzten Frameworks für das IT Service Management gilt,[[1]](#footnote-1) werden verschiedene Konzepte und Leitlinien zur praktischen Handhabung des IT Service Managements zur Verfügung gestellt.[[2]](#footnote-2) Einer der in ITIL beschriebener Prozesse stellt hierbei das Configuration Management dar.

Das Configuration Management dient der Handhabung von IT-Komponenten, wobei sichergestellt werden soll, dass die Assets ordentlich kontrolliert werden und dass korrekte Informationen über diese verfügbar sind.[[3]](#footnote-3) Das Ziel ist somit die Bereitstellung einer zentralen Informationsquelle über alle IT-Komponenten, Ressourcen und deren Abhängigkeiten untereinander. Hierfür werden Informationen vor allem über die Konfiguration der Assets und die Beziehung zwischen den einzelnen Assets in einem Configuration Management System (CMS) dokumentiert und in einer dazugehörigen Configuration Management Database abgespeichert. Die einzelnen Assets und deren Konfigurationsdetails werden hierbei anhand von Configuration Records festgehalten.[[4]](#footnote-4)

Die einzelnen Assets oder Komponenten werden als Configuration Item (CI) bezeichnet. Die unterschiedlichen CIs können hierbei stark in Komplexität, Größe und Art variieren. So kann ein CI ein komplettes System inklusive aller Hardware, Software und Dokumentationen oder auch nur eine kleine Hardwarekomponente darstellen.[[5]](#footnote-5) Um die vielen unterschiedlichen CIs besser zu strukturieren, werden unterschiedliche Configuration Item Typen definiert, welche üblicherweise alle eingesetzten Hardware und Software Komponenten, wie zum Beispiel Server oder Datenbanken, beinhalten. Jedes CI und der dazugehörige Configuration Record ist somit genau einem Configuration Item Typ zugeordnet.[[6]](#footnote-6) Die CIs werden anhand von Configuration Records, welche alle für das bestimmte CI relevanten Informationen beinhalten, in einem CMS festgehalten. Hierbei beschreibt jeder Configuration Record genau ein bestimmtes CI, wobei jedes CI ein eindeutiges Kennzeichen haben muss.[[7]](#footnote-7)

In der nachfolgenden Dokumentation ist nun detailliert der Entwurf, die Entwicklung und die anschließende Qualitätssicherung und Wartbarkeit eines CMS beschrieben. Dabei werden zunächst in den nachfolgenden Abschnitten die genauen funktionalen und nicht funktionalen Anforderungen von der Seite des Auftraggebers an das CMS betrachtet und anschließend deren Umsetzung durch das Projektteam dargestellt.

1. Aufgabenstellung des Auftraggebers

Im folgenden Abschnitt wird der Arbeitsauftrag zuerst grob und danach detailliert erläutert. Außerdem werden die Entwürfe, welche das Fundament des Projekts darstellen, vorgestellt. Dabei handelt es sich um mehrere Modelle.

* 1. Anforderungsanalyse

Nach der groben Beschreibung des Auftrags sollen nun die konkreten funktionalen und anschließend die nicht funktionalen Anforderungen an das CMS betrachtet werden.

Mithilfe des CMS sollen alle möglichen verschiedenen CI-Typen verwaltet werden können. Bei der Erstellung eines neuen CI-Typs sollen auch für diesen Typ übliche Attribute erstellt werden können, sodass nicht für jeden CI-Record die Attribute einzeln erstellt werden müssen, sondern bereits die vordefinierten Attribute des jeweiligen Typs vorhanden sind. Außerdem sollen die CI-Typen auch gelöscht werden können, wenn diese nicht mehr benötigt werden, um die Übersichtlichkeit des Systems zu erhöhen.

Um die einzelnen CIs im CMS festzuhalten soll für jedes CI ein CI-Record mit einem bestimmten CI-Typ einfach erstellt werden können. Hierbei sollen neben den bereits vordefinierten Attributen noch weitere hinzugefügt werden können, um auch alle besonderen Konfigurationen eines speziellen CIs im zugehörigen CI-Record abzuspeichern. Des Weiteren sollen die CI-Records auch im Nachhinein noch änderbar sein, damit die CI-Records bei Neuerungen schnell aktualisiert werden können und nicht neu erstellt werden müssen. Da viele CIs nach bestimmter Zeit auch ausgetauscht werden, sollen die zugehörigen CI-Records im CMS gelöscht werden können. Außerdem sollen die CI-Records einsehbar und möglichst schnell auffindbar sein, sodass man die gespeicherten Informationen möglichst schnell und einfach finden kann.

Um die Übersichtlichkeit noch weiter zu erhöhen, soll angezeigt werden, wie viele CI-Records zu einem bestimmten CI-Typ vorhanden sind. Wenn möglich sollten auch Diagramme in Form von Kreis oder Balkendiagrammen zur Veranschaulichung der Mengenverteilung erstellt werden.

Eine weitere angeforderte Funktionalität stellt das Login dar. Hierbei soll der Benutzer sich mit einem Benutzernamen und einem Passwort anmelden können und bei den richtigen Eingaben Zugang zur Software erlangen. Je nach der Berechtigung des angemeldeten Benutzers kann dieser dann unterschiedliche Tätigkeiten durchführen. Nach Beendigung der Tätigkeiten soll der Benutzer sich wieder ausloggen können, damit die Sitzung beendet wird und niemand Unberechtigtes Zugang erhält.

Die Benutzer sollen in unterschiedliche Benutzergruppen unterschieden werden, sodass verschiedene Berechtigungen vergeben werden können. Eine Benutzergruppe soll hierbei berechtigt sein, neue Benutzer anzulegen, wobei der Benutzername festgelegt und ein initiales Passwort vergeben wird, sodass der Benutzer sich erstmal mit dem vorher festgelegten Passwort anmelden kann und dieses dann anschließend ändern kann. Bei der Erstellung des Benutzers soll auch die dem Benutzer zugeordnete Berechtigung vergeben werden. Diese Rechte sollen auch später noch bearbeitbar sein und gegebenenfalls der Benutzername abgeändert werden können. Außerdem sollen Benutzer auch gelöscht werden können.

Neben den funktionalen Anforderungen gibt es auch noch die nicht funktionalen Anforderungen an das CMS, welche nun genauer betrachtet werden sollen. Dabei soll für eine möglichst gute Bedienbarkeit gesorgt werden, sodass die Benutzung möglichst intuitiv erfolgen kann und nicht anhand einer komplizierten Anleitung befolgt werden muss. Eine weitere Eigenschaft, wie die Bedienbarkeit verbessert werden kann, ist die Plausibilitätsprüfung bei den Eingaben. Des Weiteren soll die Performance des Systems optimiert werden, indem zum Beispiel die Antwortzeiten möglichst geringgehalten werden. Außerdem soll die Software wartbar und auch änderbar sein, damit diese in Zukunft angepasst werden kann, wenn es Änderungsbedarf gibt. Eine wichtige Anforderung an das System stellt auch die Sicherheit dar, da das CMS viele wichtige Daten beinhaltet, welche nur durch Berechtigte anpassbar sein sollen und nicht durch Unberechtigte manipuliert werden können.

* 1. Pflichtenheft

Nach der Beschreibung der Anforderungen an das CMS soll nun deren Umsetzung durch das Projektteam grob betrachtet und in den nachfolgenden Kapiteln vertieft werden.

Grundsätzlich wird für das Tool eine Webanwendung erstellt, in welcher die Funktionen des CMS verwendet werden können. So können hier die CI-Typen inklusive der dazugehörigen CI-Records angezeigt werden und nach einem bestimmten Namen eines CI-Records oder eines CI-Typen durchsucht werden, damit schnell das Gewünschte gefunden werden kann.

Des Weiteren können in der Webanwendung CI-Records und CI-Typen schnell angelegt werden, wobei beliebig viele Attribute definiert werden können, um alle besonderen Konfigurationsdetails zu definieren. Hier können diese auch geändert, indem neue Attribute hinzugefügt werden, oder gelöscht werden.

Zusätzlich kann ein Benutzer seine Daten ändern. Die Benutzer werden hierbei in zwei Benutzergruppen, Admin und sonstige Benutzer, unterschieden. Die Benutzer, welche als Admin angemeldet sind, können in der Webanwendung auch die Benutzer verwalten.

Die Oberfläche der Webseite wurde hierbei mit HTML erstellt und mit CSS gestylt. Aufgrund von Vorkenntnissen in der Softwareentwicklung mit Java wurde die Datenbankanbindung mithilfe von Java implementiert, wobei das Springframework zur Vereinfachung verwendet wurde, da dieses den Zugriff auf die Datenbank erleichtert und eine integrierte Datenbank anbietet. Zur Unterstützung für die Darstellung der Daten auf der Webseite wurde neben Spring auch Thymeleaf verwendet. Die Verwendung der Frameworks und die Umsetzung der im Lastenheft definierten Anforderungen wird in den nachfolgenden Kapiteln detaillierter dargestellt.

* 1. Modelle

In dem nachfolgenden Kapitel soll nun der Entwurf für das CMS anhand von verschiedenen Modellen dargestellt werden. Zuerst wird hierbei der allgemeine Phasenplan für die Durchführung der Fallstudie beschrieben. Anschließend werden vertieft die Funktionalitäten anhand eines Use-Case-Diagrammes und deren Implementierung anhand eines Klassendiagrammes betrachtet und der Aufbau der Datenbank anhand eines Entity-Relationship-Modells (ERM) erklärt.

* + 1. Use-case-Diagramm

Um die verschiedenen Funktionalitäten des CMS detaillierter darzustellen, wurde ein Use-Case-Diagramm (siehe Abb. 2) zur Veranschaulichung erstellt.

Ein Use-Case-Diagramm dient zur Darstellung, wie ein Benutzer des Systems, auch gennant Akteur, verschiedene Tätigkeiten in dem System durchführt. Ein Use-Case beschreibt hierbei den Ablauf, welcher ein Akteur bei der Durchführung einer Tätigkeit durchläuft, um ein bestimmtes Ziel zu erreichen.[[8]](#footnote-8)

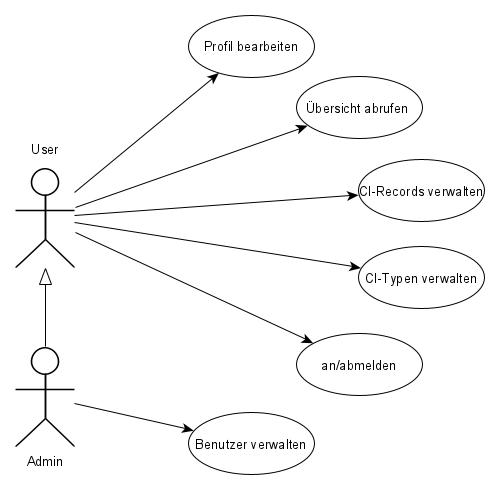


Abb. 1: Use-Case-Diagramm CMS

Als Akteure agieren in dem CMS die Benutzer, wobei die Benutzer in Admins und sonstige Benutzer unterschieden werden. Hierbei sind die Admins Benutzer mit zusätzlichen besonderen Berechtigungen. So können die Admins die gleichen und noch weitere Funktionalitäten wie die anderen Benutzer nutzen. Im Folgenden werden nun die einzelnen Use-Cases detaillierter betrachtet.

Der erste Use Case ist hierbei „Profil bearbeiten“. Jeder Benutzer kann sein Profil bearbeiten, wobei dieser seinen Benutzername und das Passwort ändern kann. Die Voraussetzung für diesen Use Case ist, dass der Benutzer bereits am System angemeldet ist. Diese Funktionalität kann genutzt werden, indem der Benutzer auf sein Profil geht und die entsprechenden Änderungen einträgt. Um das Passwort zu ändern, muss das aktuelle Passwort und das neue zweimal eingegeben werden, damit eine versehentliche falsche Eingabe eines neuen Passworts verhindert wird. Somit wäre ein sekundäres Szenario, dass das aktuelle Passwort falsch oder zwei unterschiedliche neue Passwörter eingegeben werden. In diesen Fällen wird eine Fehlermeldung angezeigt, damit der Benutzer seine Eingaben überprüfen kann. Wenn sich keine Widersprüche ergeben, werden die Daten des Benutzers aktualisiert und in der Datenbank gespeichert. Bei der nächsten Anmeldung im System kann der Benutzer sich nun mit den neuen Daten anmelden.

Ein weiterer Use Case stellt hierbei die „Übersicht abrufen“ dar. Eine Übersicht über die CI-Records und die CI-Typen kann angezeigt werden, indem in der Navigationsleiste das „Dashboard“ aufgerufen wird. Anschließend kann der Benutzer alle vorhandenen CI-Typen und die dazugehörigen CI-Records einsehen. Hierbei ist auch vorausgesetzt, dass der Benutzer am System angemeldet ist, damit niemand Unberechtigtes die gespeicherten Informationen einsehen kann.

Bei dem Use-Case „CI-Records verwalten” werden unterschiedliche Funktionalitäten zusammengefasst. So kann ein Benutzer sowohl CI-Records erstellen, bearbeiten und löschen. Bei der Erstellung muss hierbei der CI-Typ, der Name des CIs und die Werte der dazugehörigen Attribute angegeben werden. Um eine fehlerhafte Eingabe des CI-Typs zu vermeiden, kann dieser hier anhand einer Drop-down-Liste ausgewählt werden. Wie bei den vorherigen Use-Cases ist auch hier vorausgesetzt, dass der Benutzer am System angemeldet ist. Um CI-Records zu ändern oder löschen, muss der Benutzer den entsprechenden CI-Record auswählen und kann diesen dann durch das Betätigen des jeweiligen Buttons bearbeiten oder auch löschen.

Um CI-Records zu erstellen, müssen auch CI-Typen definiert werden. Dies wird durch den Use-Case „CI-Typen verwalten“ beschrieben. Um neue CI-Typen zu erstellen muss lediglich ein Name für den CI-Typ und die Namen der dazugehörigen Attribute definiert werden. Hierbei muss der Name des CI-Typs eindeutig sein, da es nicht mehrere CI-Typen mit gleichen Namen geben kann. Ein sekundäres Szenario wäre somit, dass ein CI-Typ erstellt werden soll, der schon vorhanden ist. Bei der Erstellung eines neuen CI-Typs können hierbei beliebig viele Attribute erstellt werden. Wenn CI-Typen nicht mehr benötigt werden, können diese auch gelöscht oder auch nur geändert werden. Hierfür muss der entsprechende CI-Typ ausgewählt werden und kann dann durch das Betätigen eines Buttons geändert oder gelöscht werden. Analog zu den vorher beschriebenen Use Cases muss auch hier der Benutzer angemeldet sein.

Ein weiterer Use Case, welchen alle Benutzer anstoßen können, ist das „an/abmelden“. Um Zugang zum CMS zu erlangen, müssen die Benutzer sich anmelden. Hierfür müssen diese ihren Benutzernamen und ihr Passwort im Anmeldefenster eingeben. Wenn die Eingaben stimmen, erhält dieser Zugang. Durch das Anmelden wird dafür gesorgt, dass nur Berechtigte Zugriff auf die im CMS gespeicherten Daten erhalten. Um die Sitzung zu beenden, können sich die Benutzer durch das Betätigen des Logout-Buttons abmelden. Eine Voraussetzung für das Anmelden ist hierbei, dass der Benutzer bereits registriert ist und ein initialer Benutzername und Passwort vergeben wurde. Die Voraussetzung für das Abmelden ist, dass der Benutzer in dem Moment angemeldet ist. Ein sekundäres Szenario wäre hierbei, dass der Benutzer den Benutzername oder das Passwort falsch eingibt. In diesem Fall erscheint eine Fehlermeldung und der Benutzer kann seine Eingaben nochmal überprüfen.

Die Benutzer werden hierbei durch die Admins angelegt. Dies wird durch den Use-Case „Benutzer verwalten“ beschrieben. Die Admins haben somit eine andere Ansicht als die anderen Benutzer, da diese im Profil noch die Möglichkeit haben einen neuen Benutzer zu erstellen. Bei der Erstellung eines neuen Benutzers vergeben die Admins den Benutzernamen und ein Passwort, wobei der Name des Benutzers eindeutig sein muss. Wenn also ein Benutzer mit bereits vorhandenem Namen angelegt werden soll, erscheint hierbei eine Fehlermeldung. Um einen neuen Benutzer anzumelden ist vorausgesetzt, dass der Admin im System angemeldet ist und eine entsprechende Berechtigung hat.

* + 1. Klassen-Diagramm

Nach der Funktionenerfassung anhand des Use-Case-Diagrammes (siehe Abschnitt 2.2) wurde auch ein Klassendiagramm (siehe Abb. 3) in der Unified Modeling Language (UML) zum Entwurf der Software erstellt. UML ist eine standardisierte Sprache zur Modellierung von objektorientierten Klassen.[[9]](#footnote-9)

In dem nachfolgenden Klassendiagramm werden zur besseren Übersichtlichkeit der Beziehungen zwischen den Klassen des Diagrammes nur die Klassennamen dargestellt, da die Variablen der Klassen mit den im ERM (siehe 2.4) dargestellten Attributen übereinstimmen. Die Methoden der Klassen sind sogenannte Get- und Set- Methoden, mit denen die Instanzvariablen geändert oder abgefragt werden können, da die Variablen private sind, um die Daten zu schützen.

In UML werden verschiedene Beziehungsarten zwischen den Klassen definiert. Die Assoziation wird hierbei durch eine einfache Linie dargestellt. Eine Assoziation bedeutet, dass die Objekte zweier Klassen etwas miteinander zu tun haben. Die genaue Beziehung wird anhand einer kurzen Beschreibung an der Verbindungslinie und anhand der Multiplizität definiert. Über die Multiplizität kann angegeben werden, wie viele Objekte einer Klasse an der Beziehung beteiligt sind.[[10]](#footnote-10)

Eine Verfeinerung der Assoziation stellt die Aggregation und die Komposition dar. Die Aggregation wird durch eine unausgefüllte und die Komposition anhand einer schwarz ausgefüllten Raute dargestellt. Eine Aggregation ist eine Beziehung, die eine „Ist-Teil von Beziehung“ beschreibt. Eine Komposition ist eine Aggregation mit engerer Beziehung, da dort die Lebensdauer beider Instanzen gleich ist.[[11]](#footnote-11)

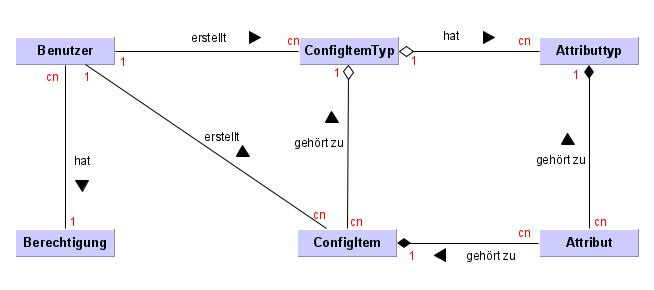


Abb. 1: Klassendiagramm CMS

Analog zum Akteur Benutzer im Use-Case-Diagramm gibt es hier eine Klasse Benutzer, welche die jeweiligen Attribute des Benutzers definiert. Da ein Admin auch ein Benutzer mit ähnlichen Eigenschaften ist, werden dessen Daten auch anhand einer Instanz der Klasse Benutzer gespeichert. Die Berechtigungsgruppen werden hierbei dann anhand der zugeordneten Berechtigungsinstanz definiert, wobei jeder Benutzer nur eine Berechtigung haben kann. Die Benutzer erstellen dann null bis beliebig viele ConfigItems und ConfigItemTypen.

Da ein ConfigItemTyp aus vielen ConfigItems besteht, handelt es sich hierbei um eine Aggregation. Allerdings kann ein ConfigItem auch nicht mehr zu einem bestehenden ConfigItemTyp zuordbar sein, wenn beispielsweise ein ConfigItemTyp gelöscht wurde, ohne dass zuvor alle dazugehörigen ConfigItems gelöscht wurden. Da die ConfigItems nicht automatisch gelöscht werden sollen, damit keine Daten verloren gehen, können diese auch noch ohne den dazugehörigen ConfigItemTyp bestehen. Deshalb handelt es sich hierbei nicht um eine Komposition, sondern um eine Aggregation.

Die vordefinierten Attribute für den ConfigItemTyp werden hierbei in einer anderen Klasse festgehalten, damit eine beliebige Anzahl an Attributen zu einem ConfigItemTyp hinzugefügt werden können. Die Attributarten werden hierbei in der Klasse Attributtyp definiert. In der Klasse Attribut wird dann einem bestimmten ConfigItem ein Wert zu einem gewissen Attributtyp zugewiesen. Zum Beispiel kann für den ConfigItemTyp „Drucker“ eine Instanz der Klasse AttributTyp mit dem Namen „Anschaffungsjahr“ erstellt werden. Wenn dann ein ConfigItem-Record des Typs Drucker erstellt wird, wird in einer Instanz der Klasse Attribut das Anschaffungsjahr des bestimmten Druckers festgehalten. Da ein Attribut genau zu einem Attributtyp und einem ConfigItem gehört und nur existiert, solange der dazugehörige Attributtyp und der CI-Record existiert, handelt es sich hierbei um Kompositionen.

Hierbei werden Attributtypen immer zu einem bestimmten ConfigItemTyp erstellt, falls unterschiedliche ConfigItemTypen einen gleichen Attributtyp besitzen, wird hierfür also ein weiterer Attributtyp mit der gleichen Bezeichnung erstellt. Deshalb handelt es sich hierbei um eine „1:n“-Beziehung, da kein Attributtyp mehreren ConfigItemTypen zugeordnet werden kann.

* + 1. Entity-Relationship-Modelle

Nach der Beschreibung des Klassendiagramms, wird in diesem Abschnitt der Entwurf für die Datenbank beschrieben. Hierfür wurden Entity-Relationship-Modelle (ERM) erstellt. Dabei wurde zwischen der Datenbank für die Configuration Items (siehe Abb. 5) und der Berechtigungsdatenbank (siehe Abb. 4) unterschieden, da nicht festgehalten wird, wer welchen CI-Record oder CITyp erstellt hat, sodass die Verbindung zwischen Benutzer und CI-Record beziehungsweise CI-Typ nicht in der Datenbank gespeichert wird.

Das ERM ist ein Modell, um Daten strukturiert darzustellen. Hierbei entspricht eine objektorientierte Klasse einem Entitätstyp des ERM, weshalb das ERM und das Klassendiagramm einander sehr ähnlich sind. Eine Entität ist ein eindeutig identifizierbares Objekt, welches sowohl konkrete Gegenstände wie Gebäude oder Personen oder auch abstrakte Konzepte wie Verträge oder Gesetze darstellen kann. Ähnliche Entitäten werden hierbei zu Entitätstypen zusammengefasst, welche verschiedene Eigenschaften haben. Diese Eigenschaften werden als Attribute bezeichnet. Im ERM werden die Beziehungen zwischen den verschiedenen Entitätstypen mit den dazugehörigen Multiplizitäten dargestellt. Die Multiplizität beschreibt, wie viele Entitäten eines Entitätstypen an der Beziehung beteiligt sind.[[12]](#footnote-12)



Abb. 2: Berechtigungsdatenbank-ERM

In der Usertabelle wird der Name und das Passwort des Benutzers gespeichert. Außerdem wird noch die BerechtigungsID als Fremdschlüssel aus der Berechtigungstabelle in der Benutzertabelle abgespeichert, da jeder Benutzer eine bestimmte Berechtigung hat. In der Berechtigungstabelle wird die Bezeichnung der jeweiligen Berechtigung festgehalten und als eindeutiger Schlüssel eine ID vergeben.



Abb. 3: CI-Datenbank-ERM

In der CI-Datenbank werden alle Informationen bezüglich der CI-Records und CI-Typen festgehalten. Beim ConfigItemTyp wird lediglich der Name festgehalten, welcher zugleich auch als Schlüssel dient. Das bedeutet, dass der Name eindeutig sein muss und nicht öfters vorkommen darf. Die dazugehörigen Attribute werden hier in der AttributTyp Tabelle mit einem Namen und dem ConfigItemTypnamen als Fremdschlüssel des dazugehörigen ConfigItemTyp gespeichert.

Ebenso hat die Tabelle, in der die ConfigItems gespeichert werden, den Namen des übergeordneten ConfigItemTyp als Fremdschlüssel gespeichert, damit zuordbar ist, zu welchem ConfigItemTyp das ConfigItem gehört. Außerdem hat das ConfigItem einen Namen und eine eindeutige ID als Schlüssel.

Der Schlüssel des ConfigItems wird als Fremdschlüssel in der Attribut-Tabelle festgehalten, sodass eindeutig zuordbar ist, zu welchem ConfigItem das Attribut gehört. Außerdem wird zur Zuordnung des Attributs zu einem bestimmten Attributtyp der Fremdschlüssel gespeichert. In der Attribut-Tabelle wird dann der Wert des Attributtyps eines bestimmten CI-Records abgespeichert.

1. Organisation

Dieser Abschnitt stellt die organisatorischen und formalen Überlegungen der Fallstudie dar. Darunter fallen neben der Projektorganisation und der Unterteilung des Projekts in Projektphasen auch die Vorüberlegungen zur eigentlichen Durchführung.

* 1. Projektorganisation

Um ein Projekt erfolgreich durchzuführen gilt es zuerst einmal eine Projektorganisation aufzustellen. Hierbei unterscheidet man grundsätzlich zwischen drei Formen der Projektorganisation:

* Reine Projektorganisation
* Stabs Projektorganisation
* Matrix Projektorganisation

Das Ziel einer Organisationsstruktur ist es das gemeinsame Rollenverständnis innerhalb des Projektteams zu unterstützen und für klare Verantwortlichkeiten und Befugnisse aller Projektmitarbeiter zu sorgen, sodass das Projekt schneller und reibungsloser verlaufen kann.[[13]](#footnote-13)

Da es sich bei der Fallstudie zum Thema Softwareentwicklung um ein Projekt handelt, welches neben dem alltäglichen Vorlesungsgeschehen bearbeitet werden soll, fiel die Wahl der Projektorganisation auf eine Stabs Projektorganisation. Bei der Stabs-Projektorganisation hat die Projektleitung, welche bei uns durch Daniel Reimann übernommen wurde, eine reine koordinierende Funktion, also keine Weisungsrechte. Der Projektleiter handelt als Auftraggeber für die anderen Abteilungen, die an dem Projekt mitarbeiten.

Um die Arbeit an der Fallstudie zu erleichtern wurde die Gruppe in zwei Expertengruppen unterteilt, die Gruppe Frontend und die Gruppe Backend. Beide Gruppen können als einzelne Abteilungen angesehen werden. Zudem hat Marcel Mahlfeld neben seiner Mitarbeit in der Abteilung Frontend noch die Organisation und Koordination der Projektdokumentation übernommen.

Die beiden Gruppen wurden nach Interessen und vor allem nach dem Wissen eingeteilt, so haben sich die jeweiligen Gruppenmitglieder, die mit dem Backend schon Erfahrungen haben, zusammengeschlossen, genauso wie die Mitglieder, die mit dem Frontend bereits Erfahrungen haben. Damit wurde bezweckt, dass die jeweiligen Gruppen so eingeteilt sind, dass jedes Gruppenmitglied etwas zum Projekt beitragen kann und seinen eventuellen Stärken bestmöglich genutzt werden können. Somit wird die Zusammenarbeit innerhalb der Gruppen erleichtert und das Projekt kann schneller und besser zum gewünschten Ziel geführt werden.

Auch innerhalb der einzelnen Untergruppen wurden die verschiedenen Aufgaben weiter unterteilt und unter den Gruppenmitgliedern aufgeteilt, somit sollte eine schnelle Erreichung des Gesamtzieles ermöglicht werden. Durch die Verteilung der einzelnen Aufgaben auf die verschiedenen Gruppenmitgliedern wurden die einzelnen Personen entlastet und die Aufgaben konnten parallel zueinander erledigt werden.

Während der wöchentlichen Treffen, die jeden Mittwochnachmittag stattgefunden haben, wurden die aktuellen Zwischenstände in Anwesenheit des gesamten Teams besprochen. In diesem Rahmen wurden neben den Zwischenständen auch aufgetretene Probleme aufgegriffen und in der Gruppe besprochen. Außerdem wurden kleine Wochenziele und weitere organisatorische Abläufe besprochen. Alle wichtigen Erkenntnisse der Gruppenbesprechungen wurden im oben angesprochenen Protokoll festgehalten, welches allen Gruppenmitgliedern in Form einer Excel-Tabelle zur Verfügung steht. Dieses Protokoll kann als zentraler Bestandteil der Projektorganisation und des Projektablaufs gesehen werden.

Die Expertengruppen trafen sich zusätzlich zu den wöchentlichen Treffen in individuellen Meetings, um ihre Aufgaben zu erledigen. Die Unterteilung erfolgte in ein Frontend- und ein Backendteam. Innerhalb des Frontend-Teams wurden die Aufgaben verteilt. Es fielen Arbeitsschritte, wie z.B. die Erstellung eines Logos, der Aufbau der Website durch die Erstellung des HTML-Quellcodes und das Design der Website durch die Implementierung des CSS-Quellcodes an. Außerdem mussten einige Funktionen durch den Einsatz der Programmiersprache JavaScript dynamisch aufbereitet werden. Diesen Aufgaben gingen die Projektmitglieder Daniel Reimann, Gianna Dihpol, Laura Biberle und Marcel Mahlfeld nach.

Auch das Backend-Team teilte die anfallenden Aufgaben auf. Die wesentlichen Aufgaben des Backends lagen darin, eine Datenbank zu entwerfen und zu implementieren, eine Geschäftslogik zu erstellen und die Individuellen Anforderungen der einzelnen Arbeitsschritte an das Frontend zu erstellen. Den Aufgaben des Backend-Teams gingen Dimitri Wagner, Pascal Jung, Johanna Gröll, Max Schneider und Til Wolmuth nach.

* 1. Git

Um kollaborativ als Team an einer gemeinsamen Code-Basis zu arbeiten und gemeinsam Code oder Software zu entwickeln, ist es sehr von Vorteil Software zu nutzen, die es möglich macht zusammen an Code zu arbeiten ohne direkt die Änderungen von den Team-Kollegen zu überschreiben. Hierfür bieten sich sehr Werkzeuge zur Versionsverwaltung an. Versionsverwaltung ist ein System, welches Änderungen an einer oder mehreren Dateien, die zusammenhängen über die Zeit hinweg protokolliert. Dadurch wird es ermöglicht, dass man später im Entwicklungsprozess die älteren Versionen einsehen kann und auch auf diese zurückgreifen kann, falls dies nötig sein sollte. Dadurch ist es im Entwicklungsprozess möglich zu überblicken an welcher Stelle Fehler oder Probleme entstanden sein konnten, um daraus viele wertvolle Schlüsse ziehen zu können und dadurch wird auch ein Rollback, also ein Zurückstellen der Software auf einen älteren Versionsstand, möglich.

Wir haben uns im Team für das Versionsverwaltungssystem GIT entschieden. Hierbei gibt es ein großes Repository, ein Ablageort für alle Dateien einer Software, das online gespeichert wird. Jedes Mitglied des Teams kopiert sich dann eine lokale Version dieses Repositorys auf seinen Rechner. Wenn man dann Änderungen vornimmt am Quellcode und zufrieden damit ist, kann man diesen neuen Stand des Quellcodes durch einen Push-Befehl in seiner Entwicklungsumbegung den derzeitigen Stand ins Repository hochladen und dieser Stand kann von den anderen Mitentwicklern durch einen Pull-Befehl auf das lokale Repository geklont werden. Dadurch wird ermöglicht, dass mehrere Teammitglieder gleichzeitig am Quellcode arbeiten können, ohne sich gegenseitig zu behindern.

* 1. Projektphasen

Für den erfolgreichen Abschluss eines Projektes, sollte dieses zunächst einmal in verschiedene Phasen unterteilt werden. Hierdurch soll für mehr Übersichtlichkeit innerhalb des Projekts gesorgt werden. Zudem wird, durch die Unterteilung des Projekts, die Organisation des Projekts vereinfacht, da somit ein zeitlicher Ablauf bereits grob vorgegeben ist und die Einteilung, der für die einzelnen Phasen benötigten Zeit, erleichtert wird. Das Projekt wurde in fünf Phasen unterteilt. Die Dokumentation des Projekts fand parallel dazu statt.

* + 1. Die Organisations- und Programmentwurfsphase

Bevor die eigentliche Arbeit an dem Projekt gestartet werden konnte, galt es zuerst einmal die Projektorganisation zu erarbeiten. Sobald die Untergruppen aufgeteilt waren, haben sich die einzelnen Gruppen zusammengesetzt und einen Programmentwurf erstellt, welcher dann in einem Treffen der gesamten Gruppe vorgestellt wurden, um schon im Voraus eventuell auftretende Probleme oder Hindernisse zwischen dem Frontend und dem Backend zu vermeiden.

* + 1. Die Erstellungsphase

Nachdem der Programmentwurf erstellt und besprochen ist, geht das Projekt über in die Erstellungsphase. In dieser Phase erledigen die beiden Untergruppen getrennt voneinander ihre für das Projekt benötigten Aufgaben. Natürlich bleibt der Austausch von Informationen und Fortschritten, in Hinsicht auf die Aufgaben, zwischen den beiden Gruppen sehr eng und ausführlich. Dies bewirkt das Probleme und Unstimmigkeiten zwischen den Aufgaben des Frontend und des Backend sehr früh schon erkannt werden und somit schnell wieder behoben werden können bevor es zu komplizierteren Problemlösungen kommen muss.

* + 1. Die Zusammenführungs- und Anpassungsphase

Nachdem die beiden Gruppen ihre Aufgaben soweit wie möglich erledigt haben, kommt es nun zur Zusammenführung der Ergebnisse. Hierbei treten kleinere Unstimmigkeiten auf, welche dann in Zusammenarbeit der beiden Gruppen meist sehr schnell und effizient gelöst werden können.

* + 1. Die Test- und Dokumentationsphase

Ist das Programm nun fertig zusammengeführt und alle Anpassungen sind erledigt, gilt es nun das Programm zu testen. Hierbei geht es darum herauszufinden, ob alle Funktionen des Programms so funktionieren, wie es sich der Auftraggeber vorstellt. Auch hier arbeiten die beiden Gruppen wieder zusammen. Ist die Funktionsweise zufriedenstellend überprüft und mögliche Fehler behoben worden, gilt es nun noch die Dokumentation zu verfassen. Für die Dokumentation werden alle Protokolle und bereits verfasste Teile der Dokumentation, die im Laufe des Projektes verfasst worden sind, zusammengetragen und zu einer Dokumentation vereint. Dadurch entsteht eine möglichst genau Wiedergabe des Projektverlaufs und eine ausführliche Erläuterung der getroffenen Entscheidungen.

* + 1. Die Fertigstellungsphase

In der letzten Phase der Fertigstellungsphase werden noch letzte Änderungen vorgenommen, damit die Wünsche des Auftraggebers so gut wie möglich erfüllt werden.

* 1. Agiles Projektmanagement

Im Rahmen des Projektes haben wir uns für ein eher agile Projektmanagement entschieden. Gründe hierfür waren vor allem, dass schnell auf neue Veränderungen reagiert werden musste, sowie schnell ein fertiges Produktinkrement erwartet wurde.

Agiles Projektmanagement zeichnet sich durch eine iterative und inkrementelle Vorgehensweise, ein selbstorganisiertes Team, sowie eine schnellstmögliche Lieferung des funktionsfähigen Produkts, aus. Darüber hinaus ist es von großer Wichtigkeit, dass das Produkt gut getestet wurde und den Kundenbedürfnissen entspricht.[[14]](#footnote-14)

Das agile Projektmanagement umfasst mehrere agile Methoden. Die bekanntesten sind dabei Kanban, Scrum oder Extreme Programming.[[15]](#footnote-15)

* + 1. Scrum

Unter Scrum versteht man eine Methode, die es Menschen ermöglichen soll, komplexe Aufgabenstellungen mit höchstmöglicher Produktivität und Kreativität umzusetzen und zu managen.[[16]](#footnote-16) Scrum wird vor allem für Softwareentwicklung verwendet. Das Scrum-Rahmenwerk beinhaltet verschiedene Regeln, an die sich das Team halten soll. Außerdem beinhaltet es vorgeschriebene Rollen, Techniken und Vorgehensweisen für eine bestmögliche Umsetzung. Besonders sind außerdem die Werte, welche Scrum verkörpert. Dazu gehören beispielsweise Mut, Fokus, Offenheit, Respekt, Transparenz, sowie Anpassung.[[17]](#footnote-17)

* + 1. Vorgehensweise

Ein Projekt wird in Scrum in kurze Zeitfenster von maximal einem Monat eingeteilt, die sogenannten Sprints. Jeder Sprint sollte dabei dieselbe Länge haben.[[18]](#footnote-18) In unserem Fall betrug die Länge jedes Sprints eine Woche, da es für uns wichtig war schnell eine fertige Funktionalität zu entwickeln.

In der Theorie soll in jedem Sprint ein potenziell fertiges Produktinkrement entstehen, welches dann in dem darauffolgenden Sprint weiterentwickelt wird.[[19]](#footnote-19)

Jeder Sprint beinhaltet die formalen Ereignisse

* Sprint Planning,
* Daily Scrum,
* Sprint Review,
* Retrospective,
* Sowie die Entwicklungsarbeit.

Unabhängig von diesen Ereignissen gibt es das Product-Backlog, welches das Team über die gesamte Projektdauer begleitet. Darunter versteht man eine Liste von allen Anforderungen an das fertige Produkt.[[20]](#footnote-20) Wichtig zu wissen ist, dass das Product-Backlog sich dynamisch, vor allem durch Feedback, weiterentwickelt. So werden Verbesserungen, sowie Fehlerbehebungen ebenfalls in das Product Backlog aufgenommen.[[21]](#footnote-21) Das Product-Backlog haben wir als Excel Tabelle umgesetzt, in welcher bei jedem unserer Meetings festgehalten wurde, was erreicht wurde, sowie was es noch zu erreichen gilt. Oftmals sind uns dabei neue Anforderungen aufgefallen, die noch nicht umgesetzt wurden, daher wurde unser Backlog oft aktualisiert.

Jeder Sprint beginnt mit dem sogenannten Sprint Planning (siehe Abb. 2), in welchem geplant wird, was in dem kommenden Sprint fertiggestellt werden muss. Es wird also ein Sprint-Ziel definiert, das es zu erreichen gilt. Dabei schaut sich das gesamte Team die Product-Backlog Einträge an und entscheidet anschließend, welche davon in diesem Sprint umgesetzt werden sollen.[[22]](#footnote-22) Diese Menge an Einträgen werden als Sprint-Backlog bezeichnet.

Ein weiteres existentielles Ereignis, ist der Daily Scrum, welcher an jedem Tag des Sprints stattfindet. In diesen festgelegten 15 Minuten plant das Team, was an diesem Tag erledigt werden soll.[[23]](#footnote-23) Wichtig ist, dass der Daily Scrum jeden Tag um dieselbe Uhrzeit und an demselben Ort stattfindet. Durch dieses Ereignis soll sich die Wahrscheinlichkeit erhöhen, das Sprint-Ziel in der vorgegebenen Zeit zu erreichen.[[24]](#footnote-24)

Üblicherweise stellt sich dabei jedes Teammitglied folgende Fragen:

* Was habe ich gestern getan?
* Was werde ich heute erledigen?
* Sehe ich ein Hindernis, das mich davon abhält?[[25]](#footnote-25)

Dieses Ereignis wurde bei uns aufgrund von Zeitmangel nicht jeden Tag durchgeführt, sondern nur jeden zweiten Tag. Anfangs des Projektes haben wir uns meist nur in den einzelnen Teams, Frontend und Backend, getroffen. Doch gegen Ende des Projektes haben wir uns meist im gesamten Team getroffen, da mehr Abstimmungen nötig waren.

Zusätzlich wird am Ende jedes Sprints ein Sprint Review durchgeführt. Dabei schaut sich das gesamte Team, sowie die Stakeholder die Ergebnisse des momentanen Sprints an. Der Product-Owner stellt dabei vor, welche Product-Backlog-Einträge fertiggestellt wurden und das Entwicklungsteam präsentiert das fertige Produktinkrement.[[26]](#footnote-26) Daher ist das Ergebnis des Sprint-Reviews ein überarbeiteter Product-Backlog, welcher die Basis für den nächsten Sprint bildet.

Zwischen dem Sprint Planning und dem Sprint Review findet die Sprint Retrospective statt. Zweck davon ist es, dem Team eine Gelegenheit zu bieten, sich selbst zu reflektieren und Verbesserungen für den kommenden Sprint zu erarbeiten.[[27]](#footnote-27)

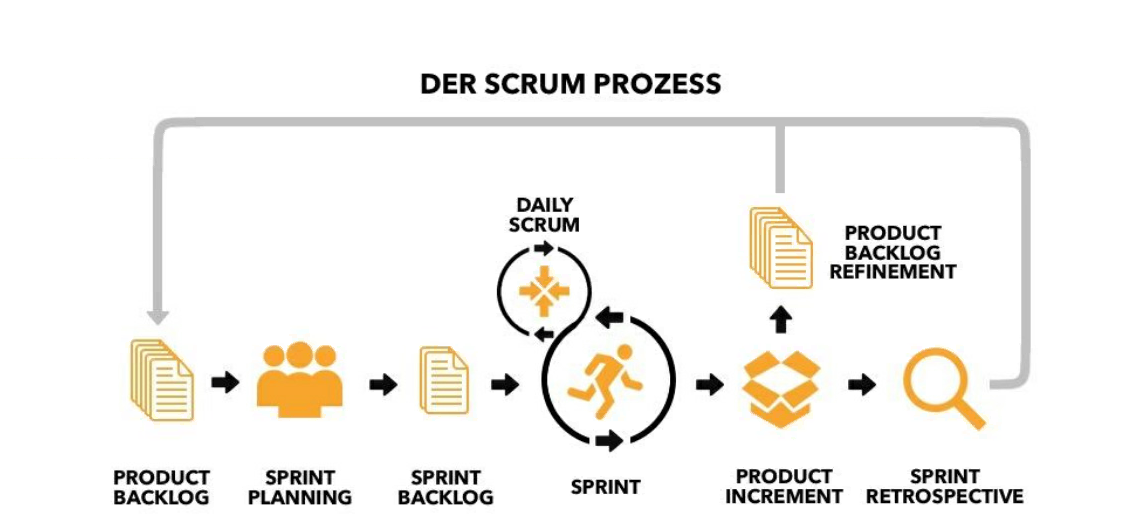


Abb. 4: Der Scrum Prozess

* + 1. Das Scrum-Team

Das Scrum Team besteht aus den Rollen Product Owner, dem Entwicklungsteam, sowie dem Scrum-Master. Als wichtigste Eigenschaften des Teams gilt, dass es sich selbstorganisiert.[[28]](#footnote-28) Daher gibt es auch keinen Projektleiter, welcher Aufgaben zuteilt. Sie bekommen vom Auftraggeber die Richtung vorgegeben, wie sie aber ihr gemeinsames Ziel erreichen bleibt ihnen selbst überlassen. Der Vorteil darin besteht, in dem das gesamte Team zu jedem Zeitpunkt den aktuellen Stand des Produktes kennt.[[29]](#footnote-29)

Der **Product Owner** ist eine Einzelperson, welche für das Product Backlog zuständig ist.[[30]](#footnote-30) Er repräsentiert die Kundensicht auf das Produkt und trifft daher die Entscheidungen, welche Eigenschaften das Produkt haben soll. Wichtig ist, er ist nicht der Projektleiter und auch nicht für die Umsetzung von Scrum zuständig.[[31]](#footnote-31) In unserem Fall war der Product Owner Daniel Reimann, welcher jedoch zusätzlich in dem Entwicklungsteam mitgearbeitet hat.

Den Kern des Teams bildet das **Entwicklungsteam**. Dieses umfasst zwischen fünf und acht Personen[[32]](#footnote-32) und ist für die Realisierung des Produktes zuständig. Zu beachten ist, dass das Entwicklungsteam aus Personen mit unterschiedlichen Fähigkeiten besteht.[[33]](#footnote-33) Dies bedeutet aber nicht, dass die Person mit der besten Expertise in einem Gebiet allein die Aufgaben dieses Fachgebiet meistern muss, sondern alle tragen zu dem Ergebnis bei. Das bedeutet es kann auch sein, dass ein Designer dem Programmierer hilft seinen Code fertig zu stellen.[[34]](#footnote-34)

Zuletzt wird von Scrum die Rolle des **Scrum-Masters** definiert. Dieser ist verantwortlich für die korrekte Umsetzung der agilen Methoden, Regeln, Werte und Vorgehensweisen.[[35]](#footnote-35) Jedoch ist er weder an der Entwicklung des Produktes beteiligt, noch hat er eine Projektleiter Funktion inne. Des Weiteren hilft er die Zusammenarbeit innerhalb des Teams weiter zu optimieren und Hindernisse zu beseitigen.[[36]](#footnote-36) Diese Rolle wurde in unser Projekt von Pascal Jung übernommen. Jedoch hat dieser trotzdem im Entwicklungsteam mitgearbeitet.

1. Programmiersprachen und Erweiterungen

Dieser Abschnitt beschreibt die im Projekt implementierten Programmiersprachen und deren genutzte Erweiterungen. Es wird nicht nur auf die reine Erläuterung der Programmiersprachen, sondern ebenfalls auf die Vorteile, die zum jeweiligen Einsatz geführt haben, eingegangen.

* 1. HTML

HTML steht für „Hypertext Markup Language“ und wird als eine textbasierte Auszeichnungssprache bezeichnet.[[37]](#footnote-37) Durch sie wird der Inhalt einer Webseite definiert und die grundlegende Struktur festgelegt. Jedoch können mit Hilfe von HTML nicht nur Inhalte strukturiert werden, sondern auch weitere Informationen einer Webseite, die sogenannten Meta-Informationen, gespeichert werden. Diese umfassen beispielsweise Angaben über den Autor und sind für Suchmaschinen, sowie für das Ranking einer Webseite relevant.

HTML kann mit jedem Standardtextprogramm verfasst werden und wird oftmals in Kombination mit CSS und JavaScript verwendet.[[38]](#footnote-38)

Der Syntax von HTML ist recht umfangreich, jedoch einfach aufgebaut und dadurch leicht zu erlernen. Ein HTML Dokument wird erstellt indem die Dateiendung in ".html" geändert wird, der Inhalt eines HTML Dokuments wird dann vom Browser interpretiert.

Jedes HTML Dokument besteht aus einem Kopf und einem Körper. Die eigentlichen HTML-Befehle werden Tags genannt und stehen innerhalb des Körpers.[[39]](#footnote-39) Die Befehle unterscheiden sich in Ihrer Bezeichnung, ihrer Funktion und können unterschiedliche Attribute, wie eine Id oder eine Klasse, mitbekommen.

Das allgemeine Grundgerüst eines HTML Dokuments sieht wie folgt aus:



Im Rahmen unserer Fallstudie haben wir uns für HTML entschieden, da HTML das heutige Standard darstellt, wenn es um die Erstellung von professionellen Webseiten geht.

Einer der größten Vorteile von HTML besteht darin, dass HTML mit allen gängigen Browsern, wie Google Chrome, Firefox oder Internet Explorer, kompatibel ist und den Einsatz von Cookies unterstützt. [[40]](#footnote-40)

Darüber hinaus sind HTML Dokumente übersichtlich durch ihren logischen und strukturierten Aufbau, wodurch die Fehlersuche und die Erstellung erheblich vereinfacht wird.

Plug-ins sind weitestgehend überflüssig. Da sich alle interaktiven Inhalte auch ohne Plugins darstellen lassen.

* 1. CSS

Im Rahmen des Projektes „zero1one“ wurde die Entscheidung getroffen das Frontend mittels CSS zu gestalten.

Cascading Style Sheets, kurz CSS, ist keine richtige Programmiersprache oder Markup-Sprache, sondern eine Stylesheet-Sprache.[[41]](#footnote-41) CSS wird zusammen mit HTML eingesetzt und legt unabhängig davon die visuelle Darstellung der verschiedenen Elemente einer Website fest. Darunter fallen beispielsweise Schriften, Layouts oder Farben.[[42]](#footnote-42) CSS greift dafür auf Sprachelemente von HTML zu.

CSS ist heutzutage der Standard, wenn es um Darstellung von Websites geht, da die Strukturierung und Programmierung von Webseiten durch CSS deutlich vereinfacht wird.

Ein essenzieller Vorteil von CSS ist, dass die Gestaltung einer Website unabhängig von dessen Inhalt bearbeitet werden kann. Denn für die Gestaltung wird eine eigene Datei mit der Kennung „.css“ erstellt. Dadurch wird die HTML Seite übersichtlicher, kürzer und einfacher zu verstehen.[[43]](#footnote-43) Durch die Verkürzung des HTML Textes und das nur einmalige Laden des CSS-Sheets, kann es zu kürzeren Übertragungsgeschwindigkeiten kommen.

Ein weiterer Vorteil ist, ein CSS-Dokument kann auf mehrere HTML-Seiten wirken. Man muss daher eine Änderung nur einmal übernehmen. Dies bietet eine große Zeitersparnis und die Möglichkeit eines einheitlichen Layouts bei vielen HTML-Seiten.

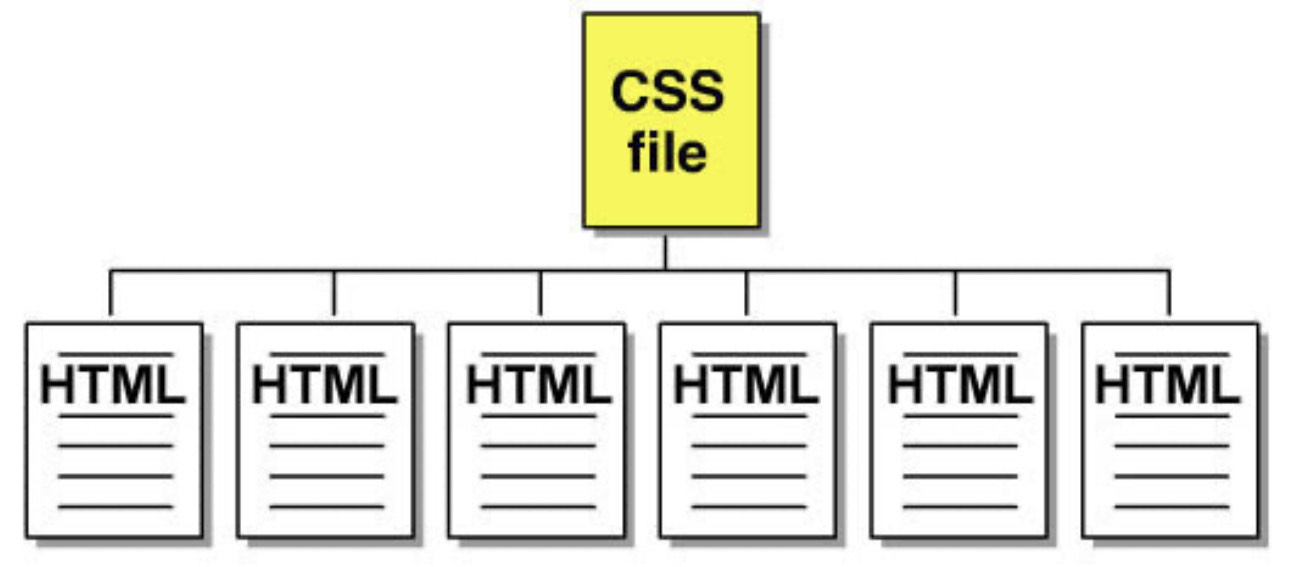


Abb. 5: CSS in Verbindung mit HTML[[44]](#footnote-44)

Darüber hinaus bietet CSS mehr Gestaltungsmöglichkeiten als beispielsweise HTML mit dem Attribut „<STYLE>“ und ist zusätzlich relativ einfach zu lernen und nachzuvollziehen.

Aus den oben genannten Gründen, sowie schon vorhandenen Erfahrungen mit CSS der einzelnen Teammitglieder, hat sich das gesamte Team für die Visualisierung mittels CSS entschieden.

* 1. JavaScript

Eine weitere Programmiersprache, die im Projekt Zero1one implementiert wurde, ist JavaScript. Zwar nimmt JavaScript nur einen geringen Anteil des Quellcodes ein, dennoch ist es ein wichtiger Bestandteil, der vom Team erarbeiteten Software. Aus diesem Grund werden im Folgenden die wichtigsten Eigenschaften und Funktionen, sowie die Vorteile, die JavaScript mit sich bringt erläutert. Dabei soll nicht nur auf allgemeine Fakten, sondern zudem auf Vorteile, die den Einsatz von JavaScript während der Durchführung der Fallstudie rechtfertigen.

JavaScript ist eine weit verbreitete clientseitige Programmiersprache. Das bedeutet, dass die Programme im Webbrowser ausgeführt werden. Es wird oft genutzt, da es von den meisten gebräuchlichen Browsern unterstützt wird.[[45]](#footnote-45) Dabei ist es eine plattformübergreifende, objektorientierte Skriptsprache. Skriptsprachen unterscheiden sich von konventionellen Programmiersprachen dadurch, dass sie nicht kompiliert werden, sondern interpretiert. Dazu ist ein Interpreter nötig, der den Quellcode bei jeder Ausführung immer wieder neu übersetzt.

Im Projekt Zero1one wird JavaScript wie oben erwähnt nur stellenweise verwendet. Auf der Seite „Hilfe“ können Informationen über die Website dynamisch abgefragt werden. Dazu ist die Datei hilfe.js notwendig. Diese ermöglicht das Ein- bzw. Ausklappen einzelner Paragraphen. Nähere Informationen dazu sind im Kapitel der Erklärung des Quellcodes zu finden.

Ein ausschlaggebender Vorteil, der für einen Einsatz von JavaScript spricht, ist die Verknüpfung mit einer HTML-Datei. Ohne große Aufwände kann der Quellcode einer HMTL-Datei, mithilfe einer Schnittstelle um eine JavaScript-Datei erweitert werden[[46]](#footnote-46). Dazu ist nur eine Zeile Code notwendig.

Einen weiteren Vorteil von JavaScript stellt die Kernfunktion der Programmiersprache dar. Objekte aus statischen HTML-Seiten können dynamisch um Eigenschaften und Methoden erweitert werden. Dadurch kann eine Website interaktiv gestaltet werden.

* 1. Java

Java ist eine objektorientierte Programmiersprache, die sich auf Grund ihres Fokus auf Objektorientierung sehr für den Einsatz in der verteilten Umgebung des Internets eignet. Mithilfe von Java ist es möglich, komplette Anwendungen zu entwickeln, die dann entweder auf einem einzelnen Computer lokal laufen können oder verteilt zwischen Servern und Clients in einem Netzwerk ausgeführt werden. Die Sprache bietet sich außerdem dazu an, kleine Anwendungsmodule oder Applets als Teil einer Website zu bauen. Diese Applets bieten sich sehr gut an, um dem Nutzer die Möglichkeit zu geben mit der Website zu interagieren. Da Java-Applets auf fast jedem Betriebssystem ohne Neukompilierung laufen und weil Java keine betriebssystemspezifischen Erweiterungen oder Variationen hat, wird Java im Allgemeinen als die strategischste Sprache angesehen, in denen man Anwendungen für das Web entwickeln sollte. Allerdings kann auch JavaScript für sehr kleine Anwendungen, die auf dem Web-Client oder Server ausgeführt werden, sehr nützlich sein.

Die Erklärung Javas zeigt, dass sich Java sehr für unser Vorhaben in diesem Projekt anbietet, da es unser Ziel ist, eine Webanwendung zu entwickeln, die in einem serverseitigen, verteilten Netzwerk läuft. Es stellt also eine gute Möglichkeit dar, für uns die Systeme, die auf das Netzwerkt verteilt sind, miteinander kommunizieren zu lassen. Außerdem bietet sich Java sehr für uns an, da Java relativ problemkompatibel mit anderen Komponenten ist. Das wichtigste Argument für den Gebrauch von Java, neben der Tatsache, dass es sich wegen seiner objektorientierten Natur gut anbietet, ist die Tatsache, dass wir seit ca. drei Semestern viel mit Java arbeiten und alle Grundlagen mittlerweile kennen. Außerdem wurde uns sogar im Kurs „Datenbanken“ beigebracht Datenbanken in SQL mit Java zu verbinden und zusammen zu nutzen. Wenn man dies mit den Anforderungen an das Projekt vergleicht, sieht man eigentlich direkt den Grund warum wir uns für Java entschieden haben.

* 1. Thymeleaf[[47]](#footnote-47)

„Thymeleaf ist eine modernes serverseitig betriebene Java-Template-Engine für Web-Anwendungen sowie alleinstehende Umgebungen.“[[48]](#footnote-48) [übersetzt aus dem Englischen] ist die Definition der offiziellen Thymeleaf Homepage und wird benutzt zur Verarbeitung und Erstellung von HTML, XML, JavaScript, CSS und Text. Die Bibliothek von Thymeleaf ist außerdem äußerst erweiterbar und durch ihr natürliches Template-Wesen gewährleistet Thymeleaf, dass Vorlagen ohne Backend als Prototyp getestet werden können. Da Thymeleaf Module fürs SPRING-Framework, die Möglichkeit viele der oft benutzten Tools zu integrieren, sowie seine eigenen Funktionalitäten einzubinden, anbietet, bietet es sich sehr gut für die moderne HTML5-Webentwicklung an.

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. Lediglich auf Seiten mit statischem Inhalt, wie z.B. die Hilfe, war der Einsatz nicht nötig. Generell 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 sogenannte „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 demselben 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 6 Schnittstellen).

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. Spring

Spring ist ein weitverbreitetes Framework, dessen Entwicklung momentan von Pivotal Software vorangetrieben wird. Es ist für Java-Anwendungen konzipiert und ist das größte und mächtigste Framework für die Programmiersprache Java, das man zurzeit auf dem Markt finden kann. Im Jahr 2002 wurde die erste Version des Spring Frameworks als Open Source Projekt veröffentlicht.

Die Intention hinter Spring ist zum einen die Entwicklung mit Java zu vereinfachen. Oft steht man am Anfang eines Projekts und hat noch nichts, auf das man aufbauen oder anknüpfen kann. Dabei haben die meisten Projekte in ihren Grundzügen ähnliche Aufgaben. Für diese gleichbleibenden Aufgaben bietet sich der Einsatz eines Frameworks, diese bildet ein Rahmengerüst für den Programmaufbau an, der genau jene sich wiederholenden Aufgaben bereits implementiert.

Die andere große Intention hinter Spring ist es saubere und gute Programmierpraktiken zu unterstützen und zu fördern. In großen Programmen geht schnell die Übersichtlichkeit verloren. Durch einen Einheitlichen Aufbau mit Spring ist es oft möglich auch bei großen und komplexen Programmen noch den Durchblick zu behalten.

Spring-Projekte nutzen zur Projektverwaltung Apache Maven oder Gradle. In der Fallstudie wird mit Apache Maven gearbeitet. Maven ist ein Build-Management-Tool, das in Java geschrieben wurde, um Java-Projekte zu verwalten. Es wurde von der Apache Software Foundation entwickelt und ist ein Open-Source-Projekt. Maven erleichtert das Bauen, Verpacken und Testen von Anwendungen. Zudem bietet Maven eine Verwaltung der Abhängigkeiten zu anderen Projekten und Bibliotheken in einer zentralen Datei, der sogenannten pom.xml. In ihr werden auch wichtige Eigenschaften des Projekts definiert. In der POM werden neben der Artefakt-ID und der Group-ID, die zur eindeutigen Identifizierung dienen, auch Parameter definiert, wie das Dateiformat, in das die Anwendung verpackt werden soll.

Das Grundkonzept von Spring beruht auf der Inversion of Control und der Dependency Injection. Die Inversion of Control beschreibt ein Umsetzungsparadigma, das häufig in der objektorientierten Programmierung Anwendung findet. Anstatt das ein Programm alle seine Dienste steuert und so nur die Standardfunktionen benutzt, wird die Steuerung der Ausführung Unterprogrammen des Frameworks überlassen. Dieses aufgerufenen Unterprogramm muss jedoch zuvor sich und seine Funktion bei der Standardbibliothek registrieren, um aufgerufen werden zu können.

Das Prinzip der Dependecy Injection ist eine Spezifizierung der Inversion of Control. In einem klassisch aufgebauten objektorientierten Programm kümmert sich jedes Objekt selbstständig um seine Abhängigkeiten. Es ist selbst verantwortlich für den Aufbau des Abhängigkeitsnetzes zu den Objekten und Ressourcen, die es benötigt. Dies bedeutet in der Praxis, das Objekt muss seine Umgebung kennen und andere Objekte instanziieren. Diese Art der Arbeit gehört aber normalerweise nicht in den Aufgabenbereich des Objekts, dessen Aufgaben haben oft nichts damit zu tun. Aus diesem Grund teilt Spring die einzelnen funktional abgegrenzten Bereiche in unterschiedliche Komponenten, die Beans. Diese stellen die Unterprogramme dar, die aufgerufen werden können. Damit die Lebenszyklen der einzelnen Komponenten so weit wie möglich auseinandergehalten werden können, werden die Abhängigkeiten im Spring-Anwendungskontext registriert und verwaltet. Wenn zur Laufzeit eine Bean nun eine Abhängigkeit fordert, wendet diese sich an den Spring-Anwendungskontext. Dieser erzeugt daraufhin eine separate Entität, die dann in die richtige Bean injiziert wird. So bleiben die einzelnen Komponenten relativ unabhängig voneinander, aber es wird trotzdem die gesamte Anwendungsfunktionalität gewährleistet.

* + 1. Gründe für die Nutzung von Spring

Aus bereits genannten Gründen bietet Spring einige Vorteile mit sich. Die Struktur wird einfach und übersichtlich gehalten. Wenn man eine standalone Anwendung entwickeln möchte, muss man zusätzlich weitere Aspekte beachten. Hier bietet Spring umfangreiche Unterstützung. Zum Erstellen einer eigenständig lauffähigen Spring-Applikation bietet Spring die Erweiterung Spring Boot an. Im Wesentlichen stellt Spring Boot lediglich eine Vereinfachung der Konfiguration einer Spring-Anwendung dar. Mit Hilfe des Online-Tools Spring Initializr kann man recht einfach ein Projekt erstellen. Dort kann man zunächst die grundlegenden Eigenschaften festlegen. Zu diesen zählen die Programmiersprache, mittlerweile unterstützt Spring neben Java auch Kotlin und Groovy, die Projektverwaltung, die Version von Spring, so wie Projektmetadaten. Zur Projektverwaltung unterstützt Spring Maven und Gradle. Wie oben bereits erwähnt wurde, wird in der Fallstudie das Build-Management-Tool Maven verwendet. Zu den Projektmetadaten gehören wichtige Informationen wie die Group-Id, die Artefakt-Id und der Name des Projekts. Es kann auch eine Beschreibung des Projekts angelegt werden. Wahlweise kann auch der Package Name geändert werden, was im Allgemeinen wenig Sinn macht, da dieser sich meist aus der Group- und der Artefact-Id zusammensetzt. Eine weitere wichtige Entscheidung stellt das Packaging Format dar. Je nach Anwendungszweck kann das Java-Projekt als War oder Jar verpackt werden. All diese Informationen können bei Bedarf auch noch zu einem späteren Zeitpunkt in der pom.xml geändert werden. Im Initializr wird auch die Version von Java festgelegt, mit der gearbeitet werden soll. Momentan stehen die Versionen 8, 11 und 15 zur Auswahl. Da mit Java 9 einige Features dazu gekommen sind, auf die wir in der Entwicklung nicht verzichten wollten, war klar, dass die Version eine neuere als Java 8 sein sollte. Da Java 15 noch frisch und neu ist und noch kaum jemand Erfahrungen mit den neusten Änderungen hat, fiel die Entscheidung auf Java 11. Hierzu gibt es im Team der Fallstudie ausreichende Erfahrungen und auch umfangreiche Dokumentation mit Beispielen und Erklärungen im Internet. Zum Konfigurationszeitpunkt können auch schon Abhängigkeiten hinzugefügt werden. Dies ist aber nicht zwingend erforderlich. Auch diese können im Laufe der Entwicklung in der pom.xml verändert und gepflegt werden.

Hat man schlussendlich das Projekt nach den eigenen Wünschen konfiguriert, kann man das Projekt erzeugen und anschließend als zip-komprimierten Ordern herunterladen. Nach dem Entpacken kann man das Projekt direkt in der Enwicklungsumgebung seiner Wahl importieren und ist startklar zum Entwickeln.

Spring Boot greift einem nicht nur bei der Initialisierung unter die Arme. Auch viele andere Konfigurationen werden größtenteils automatisch vorgenommen. Da wir uns im Projekt für eine HTML-Oberfläche entschieden haben, bietet es sich an mit Spring MVC zu arbeiten. Hierzu wird lediglich die Abhängigkeit zum „spring-boot-starter-web“ benötigt. Diese bringt nun eine Konfiguration mit, mit der der integrierte Tomcat-Server automatisch beim Start betrieben wird. Auch für den Datenbankzugriff kann Spring Boot Konfigurationen übernehmen. Erkennt Spring beim Scannen der pom.xml, dass eine Abhängigkeit zu einem Datenbanktreiber für JDBC existiert, übernimmt Spring im Folgenden die nötigen Einstellungen, damit eine Verbindung zu einer Datenbankaufgebaut werden kann.

* + 1. Erweiterungen von Spring

Damit Spring auch im Projekt der Fallstudie seine Konfigurationen richtig erstellen kann, benötigen wir einige zusätzlichen Abhängigkeiten. Folgende Spring Boot Abhängigkeiten wurden von uns eingebunden:

* spring-boot-starter-web
* spring-boot-startet-data-jpa
* spring-boot-starter-thymeleaf
* spring-starter-starter-security

Im Folgenden werden kurz die Aufgabengebiete der einzelnen Abhängigkeiten erläutert. Eine genaue Beschreibung der Anwendung erfolgt im Teil, in dem der Aufbau des Backends beschreiben wird.

Die starter-web-Abhängigkeit bringt alle grundlegenden Funktionen für unsere Anwendung mit. Dazu gehören neben Spring Core, Spring MVC mit dem eingebundenen Servlet Container Tomcat und Logging-Funktionen mit logback oder slf4j. Spring Core stellt die bereits beschriebenen Grundfunktionen von Spring zur Verfügung. Das Logging wird in unserer Anwendung nicht verwendet. Spring MVC unterstützt uns beim Bereitstellen eines Webservices. MVC konfiguriert nicht nur den Servlet Container Tomcat. Des Weiteren erzeugt und verwaltet Spring MVC auch das Dispatcher Servlet. Dadurch handelt Spring eingehende Anfragen automatisch und gleicht diese mit dem manuell erstellten Mapping ab. Bei Fehlern bietet Spring MVC auch eine automatische Fehlerseite, die gemäß dem http-Standard den Fehlercode ausgibt. Diese Funktion wird bei einer komplexeren Anwendung oder bei einer Anwendung, die produktiv gesetzt wird, höchstwahrscheinlich überschreiben.

Die startet-data-jpa-Abhängigkeit ist ein wichtiger Bestandteil der Anwendung, da durch sie ein sehr einfacher und übersichtlicher Datenbankzugriff ermöglicht wird. Spring Data JPA baut auf die bestehenden Funktionen der Java Persistence API (JPA) auf. Dadurch, dass Spring einen hohen Teil an automatischen Konfigurationen durchführt, gibt es in einem Spring Data JPA-Repository keinen unnötigen Code. Im Idealfall benötigt man nicht einmal SQL-Code. Spring kann aus gewissen Schlagworten automatisch SQL-Abfragen generieren.

|  |
| --- |
| public interface UserRepository extends JpaRepository<User, String> |

Ein Repository wird als Interface implementiert, das von einem JPA-Repository erbt. Das JPA-Repository ist generisch und wird immer von dem Typ angelegt, den es mit der Abfrage befüllen soll. Der zweite Typ ist derselbe wie der Key der Tabelle, auf der abgefragt werden soll. Der erste Typ, in diesem Beispiel der User, stellt mit seinen Attributen eine Zeile aus der Tabelle dar. Aus diesem Grund trägt die Klasse User die Annotation @Entity. Das Schlüsselattribut muss zusätzlich mit @Id annotiert werden, damit Spring die Klasse auf die Datenbank mappen kann. Ein Repository verfügt standardmäßig viele Funktionen, wie findAll(). Mit dieser Methode wird automatisch eine SQL-Abfrage generiert:

SELECT \* FROM User

Als Ergebnis liefert diese Methode eine Liste des Typs User. Möchte man Querys personalisieren, kann man einfache Abfragen als Methoden vorgeben. Wenn man einen User anhand seiner ID finden möchte, erstellt man eine Methode: findByID(String id). Daraus generiert Spring die Abfrage:

SELECT \* FROM User WHERE ID=id

id steht hier als Platzhalter für den übergebenen Wert. Mit dem in der pom.xml hinterlegten Datenbanktreiber kann in Spring so eine Verbindung zu einer Datenbank aufbauen und eine Abfrage auf dieser absetzen, ohne dabei eine Zeile SQL zu schreiben.

Die Thymeleaf Starter-Abhängigkeit bietet eine Einbindung der Tempalting Engine Thymeleaf. Diese wird in ihrem eigenen Abschnitt genauer beschreiben.

Auch für die einfache Implementierung einer Authentifizierung bietet Spring eine Erweiterung. Mit dem Spring Security Starter fordert Spring automatisch eine Authentifizierung für alle Pfade, die vom Tomcat-Servlet bedient werden. Benutzer und Passwort können statisch in der application.properties hinterlegt werden. In diesem Fall bietet Spring ein einfaches Login-Interface. Man kann das Login auch abändern und die Verifizierung selbst übernehmen. In diesem Fall müssen aber mehrere Beans überschrieben werden. Dies wird ausführlicher in der Umsetzung beschrieben.

Das Spring Framework bietet ein breites Spektrum an Funktionalitäten, mit denen für beinahe jeden Anwendungsfall eine passende Unterstützung bereitgestellt. Folglich ist Spring auch für die Anwendung der Fallstudie das geeignete Werkzeug.

1. Datenbanken

Innerhalb der technischen Anforderungen des Auftraggebers wurde eine Datenbank zur programminternen Datenverwaltung vorausgesetzt. Dabei soll dieses System die anfallenden Datenmengen persistent speichern und die Konsistenz der Nutzdaten einer Institution gewährleisten. Zudem soll dadurch im Programm auf die darin gespeicherten Daten zugegriffen werden und diese in bedarfsgerechten Darstellungsformen präsentieren.

Um die richtige Datenbank auszuwählen sollte zunächst ein genauerer Blick in den Aufbau und die Funktionen einer solchen Datenbank geworfen werden und diese dann folglich mit den im Projekt benötigten Anforderungen abgeglichen werden. Im Grunde besteht eine Datenbank aus genau zwei Teilen einmal einer Verwaltungssoftware, welche als Datenbankmanagementsystem (kurz. DBMS) bezeichnet wird. In dieser wird intern die strukturierte Speicherung der benötigten Daten organisiert und alle lesenden und schreibenden Zugriffe auf die Datenbank kontrolliert. Hierdurch können der Anwender bzw. der Entwickler die gesamte Datenbank mit Hilfe einer Datensprache vorerst einmal aufrufen und Daten anlegen bzw. diese bearbeiten oder löschen. Zudem legt ein solches Datenbankmanagementsystem das Datenbankmodell fest und entscheidet maßgeblich über die Funktionalität, sowie die Geschwindigkeit des benutzten Systems.[[49]](#footnote-49) Der Zweite Teil der Datenbank stellen die Menge der zu verwalteten Daten, der Datenbank (DB) an sich, dar. Dieser stellt sich als logisch zusammengehörender Datenbestand dar und wird von einem laufenden Datenbankmanagementsystem verwaltet.[[50]](#footnote-50) Eine solche Software zur Verwaltung der im Programm zu speichernden Daten wird in diesem Projekt benötigt. Die zuvor erwähnten Funktionen eines Datenbankmanagementsystems sind jedoch auch abgesehen von der Speicherung, der Überschreibung und der Löschung von Daten im Programm noch eine Menge andere. Dazu gehören die Verwaltung dieser Metadaten, eine Vorkehrung zu der Datensicherung, welche eine große Rolle für die in den Anforderungen verlangte Benutzerverwaltung darstellt. Zudem sind die weiteren Funktionen noch Vorkehrungen zur Datenintegrität und eine Ermöglichung des Mehrbenutzerbetriebs, welche in diesem Projekt von essentiellem Wert für die gerechte Verwaltung der Datenbank sind.



## Auswahl des Datenbankmodells

Um nun herauszufinden, welche Datenbank für die Software die richtige ist, beschäftigte sich der Backend-Teil des Teams während der Anforderungsanalyse mit den verschiedenen Arten der Datenbanken. Dabei wurden folgende Datenbanken Arten angesehen:

### **Hierarchische Datenbank**

In diesem Datenbankmodell stehen die verschiedenen Objekte ausschließlich in einer Eltern-Kind-Beziehung. Das ist so, da dieses Modell versucht die reale Welt durch eine hierarchische Baumstruktur abzubilden. Jeder Satz hat hier genau einen Vorgänger, lediglich der Anfang dieses Datenbankmodells die sog. Wurzel hat keinen Vorgänger. Die Daten werden in einer Reihe von Datensätzen darin gespeichert, diese sind folglich mit weiteren verschiedenen Feldern verknüpft. Die Verknüpfungen in einem solchen Modell werden als Eltern-Kind-Beziehung oder auch Parent-Child-Relationship genannt, da es dem bekannten Familienstammbaum sehr ähnlich ist. Nachteilig ist jedoch, dass in einer solchen hierarchischen Datenbank lediglich mit einem solchen Baum umgegangen werden kann. Zudem können keine Verknüpfungen zwischen verschiedenen Bäumen oder über mehrere Ebenen des Baumes getätigt werden.[[51]](#footnote-51)

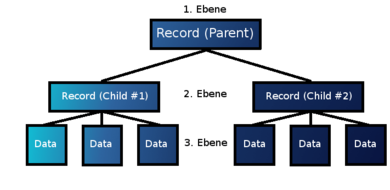


Abb. 6: Hierarchische Datenbank[[52]](#footnote-52)

### **Vernetzte Datenbank**

Hierfür werden die Daten Instanzen miteinander in einem Netz verbunden. Dieses Datenbankmodell erfordert keinerlei strenge Hierarchie, weshalb m zu n Beziehungen ohne Probleme abgebildet werden können und somit ein Datensatz mehrere Vorgänger haben kann. Um einen bestimmten Datensatz zu suchen gibt es mehrere verschiedene Wege, weshalb sich die Suche nach einem bestimmten Datensatz etwas schwer darstellt.[[53]](#footnote-53)

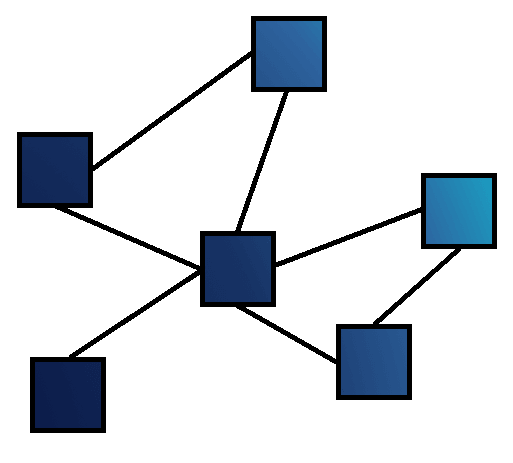


Abb.7: Vernetzte Datenbank[[54]](#footnote-54)

### **Objektorientierte Datenbank**

## In diesem Datenbankmodell werden die Beziehungen zwischen verschiedenen Datenobjekten vom Datenbanksystem selbst und Daten werden hier speziell als Objekte verwaltet. Dabei kann eine Vererbung von Eigenschaften und Daten anderer Objekte stattfinden. Ein bestimmtes Objekt modelliert hier im Normalfall einen gewissen Gegenstand und enthält somit die jeweilig essentiellen Attribute. Somit wird bei einer Anlage eines Haus-Objektes die Farbe, die Größe und der Standort als Attribute dieses Objektes gespeichert. Dazu werden jedoch auch die dazu aufrufenden Methoden für die jeweiligen Attribute mitgespeichert, wie es mit den „getter“ und „settern“ in der Objektorientierung in Java bekannt ist. Vorteile einer solchen Datenbank sind zum einem, dass dieses Modell Lücken schließt, welche bei der Programmierung moderner Datenbankanwendungen entstehen können, wenn sie in einer objektorientierten Programmiersprache entwickelt wurde. Und zudem wird das Problem der Objektidentität gelöst. Jedoch ist der Nachteil, dass für dieses Modell notwendige Schnittstellen und Tools, wie JDBC oder Spring, nicht für den Einsatz mit diesem Modell vorbereitet sind.[[55]](#footnote-55)

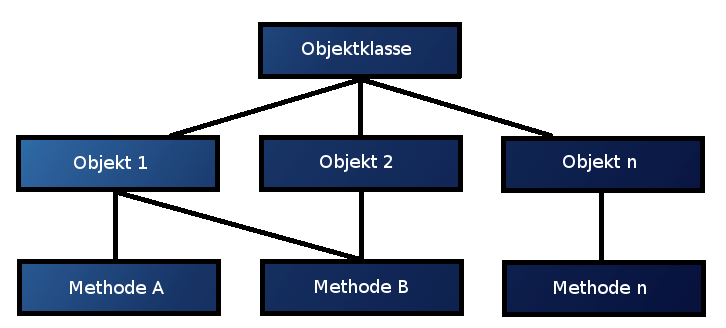


Abb. 8: Objektorientierte Datenbank[[56]](#footnote-56)

### **Relationale Datenbank**

Die in relationalen Datenbanken gespeicherten Daten werden in dafür erstellten Tabellen verwaltet. Es kann hier verschieden beliebige Beziehungen zwischen den Daten erfolgen, da diese durch Werte bestimmter Tabellenspalten festgelegt werden. Vorteilhaft hierbei sind die Anwendung von „Joins“ zum Verknüpfen von mehreren Datenbank Tabellen. Zudem sind mehrere zusätzliche Table-Funktionen, wie das addieren, subtrahieren, dividieren oder multiplizieren, bis hin zum aufsummieren möglich.[[57]](#footnote-57)

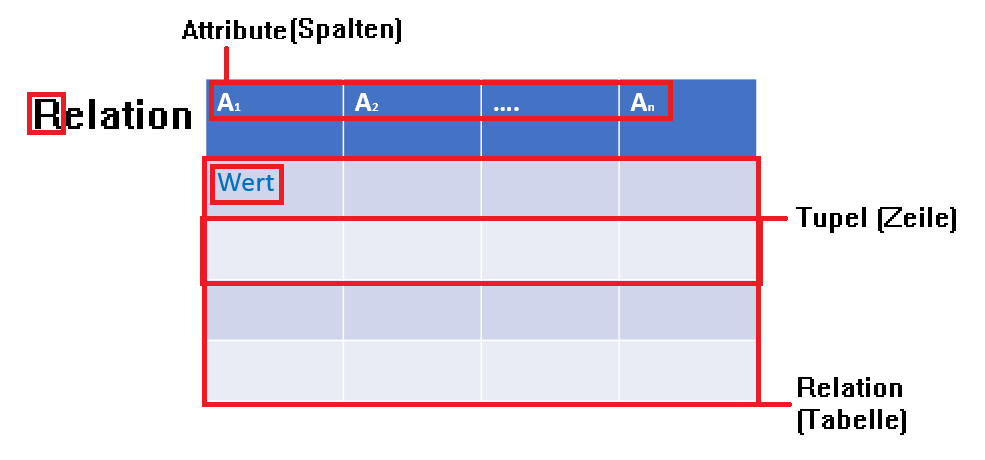


Abb. 9: Relationale Datenbank[[58]](#footnote-58)

Nach der Absprache im Team wurde sich dazu entschlossen eine Relationale Datenbank im Projekt mit einzubinden. Der Grund dafür ist, dass eine tabellarische Abspeicherung der Daten zu den Anforderungen der Anforderungsanalyse am besten passen. Zudem lässt sich mit einer solchen Datenbank ein relativ einfaches Konzept des Datenbankaufbaus erstellen. Indem mehrere Tabellen der Datenbank verbunden werden können, wird die Verwaltung und vor allem auch die Suche und folglich die Ausgabe der gewünschten Daten vereinfacht.

### **Aufbau der Datenbank**

Wie in Kapitel 2.3.3 beschrieben wurde zur Darstellung der Datenbank ein Entity-Relationship-Modell erstellt. Dies wurde gewählt, da dadurch ein genauer und einfacher Überblick über die Bestandteile der Datenbank aufgezeigt wird. Dazu wurde zwischen den Berichtigungen der User im Login und den Daten der gespeicherten ConfigItems unterschieden.



Abb. 10: Berechtigungsdatenbank-ERM

Um einen Login in der relationalen Datenbank des Projektes zu realisieren, wurden die Tabellen „User“ und die Tabelle „Berechtigung“ erstellt. Die Berechtigungen sind in der Berechtigungs-Tabelle vorgefertigt. Diese sind „Admin“ und „Benutzer“ (Berechtigungstype vielleicht verbessern). Durch diese kann im Programm unterschieden werden, wer welche Berechtigung hat und folglich welche Maßnahmen dieser auf der Oberfläche tätigen kann. Bei der Erstellung eines neuen Users schreibt das Programm in die User-Tabelle den jeweiligen Namen, das Passwort in Form eines Hash und die Referenz auf die Berechtigungs-ID der Person als Fremdschlüssel. Dabei kann jedoch jeder User nur eine einzige Berechtigung besitzen und dadurch auch nur die mit der Berechtigung verbundenen Tätigkeiten im Programm ausführen. Somit kann der Benutzer sich nach dem Starten der Anwendung im Login Fenster mit seinen Login-Daten anmelden und gelangt dadurch auf die eigentliche Anwendungsoberfläche.



Abb. 11: CI-Datenbank-ERM

Nachdem der Anwender sich angemeldet hat, wird dieser voraussichtlich CI-Typen verwalten wollen, dazu müssen die Daten dieser gespeichert werden. Hierfür hat das Backend-Team einen besonderen Aufbau der Datenbank entwickelt. Vom Anwender können Configitemtypen erstellt werden, wozu lediglich der Name dieses in der Tabelle „ConfigItem“ gespeichert wird. Zu jedem Cofigitemtyp werden verschiedene „Attributtypen“ definiert und später einmal beladen. Als Beispiel werden bei der Erstellung eines Configitemtyp mit dem Namen Handy zusätzlich die Attributtypen Marke, Speicher und Bildschirmgröße definiert. Dazu werden jeweils in der Tabelle „AttributTyp“ der Name des Attributtyps und der dazugehörige ConfigItemtypname als Fremdschlüssel gespeichert. Diese können durch den Fremdschlüssel genau einem ConfigItemtypen zugeordnet werden. Werden nun vom Anwender eine bestimmte Instanz eines Configitemtyps erstellt, werden die jeweilige ID der Instanz, der ConfigItemtypname als Fremdschlüssel und der Name der Instanz gespeichert. Durch die Speicherung des Fremdschlüssels ist es möglich die Daten später einem gewissen ConfigItemtypen zuzuordnen, weshalb es sich hier um eine 1 zu cn Beziehung handelt. Um nun noch die Daten des ConfigItems verwalten zu können, werden zu jeder Instanz eines ConfigItemtyps die definierten Attributtypen in der Tabelle „Attribut“ gespeichert. Hierfür werden die ConfigItemID und die AttributTypID jeweils als Fremdschlüssel übergeben, um einen Attributwert eines ConfigItems den Attributtypen des ConfigItemTyps zuzuordnen, weshalb es sich bei beiden um eine 1 zu cn Beziehung handelt. Zudem wird der Wert des Attributes in dieser Tabelle gespeichert, welcher somit nun über die jeweiligen Fremdschlüssel abrufbar ist.

### **Auswahl des Datenbankmanagementsystems**

Nachdem entschieden wurde eine relationale Datenbank zu nutzen, sollte nun zum Zeitpunkt des Projektentwurfes ausgewählt werden, welches Datenbankmanagementsystem genutzt wird, um nun die Datenbank im Programm zu verwenden. Dazu wurde zunächst entschieden eine MySQl Datenbank in das Programm mit einzubauen, da diese eine sehr gute Integration von verschiedenen Systemen mit sich bringt und mit einer sehr guten Performance ausgelegt ist.[[59]](#footnote-59) Dieses ist das meist verbreitete Open-Source-Software relationale Datenbankverwaltungssystem. Um diese zu Nutzen hat jeder des Backend-Teams die Anwendung XAMPP heruntergeladen. Dieses Programm ermöglicht es dem Team einen Apache Webserver und eine MySQL Datenbank auf dem Rechner zu installieren und zu konfigurieren. Damit mit dieser gearbeitet werde konnte, wurde vom Backend-Team über den mitgelieferten PHP-Admin verschiedene Testdaten in die Datenbank importiert. Diese Datenbank wurde vom Programm angesprochen und die darin vorgespeicherten Testdaten wurde auf den ersten Oberflächentests ausgegeben. Im Zuge der Entwicklung und den Blick auf die Auslieferung des Programms entschied sich jedoch das Team statt einer MySQL-Datenbank eine H2-Datenbank in das Programm mit einzubauen. Diese Datenbank ist ohne eine Installation auf jedem System lauffähig, was auf ein sogenanntes „Rapid Prototyping“ zurückzuführen ist. Zusätzlich unterstützt diese einen „Embedded Mode“ als auch einen „Server Mode“.[[60]](#footnote-60) Jedoch sind die Hauptargumente warum sich das Team für eine H2-Datenbank entschieden haben und somit keine MySQl-Datenbank verwenden noch einmal andere. Die Gründe sind nämlich zum einem, dass nun die im Programm benutze Datenbank nicht extern vom Programm aufgerufen werden muss, wie es bei einer MySQL-Datenbank der Fall ist. Sondern die Datenbank sich mitimplementiert im Programm befindet und somit mit dem Programm gleichzeitig ohne Probleme im Arbeitsspeicher läuft, was mit der geringen Speicheranforderung des Datenbanksystems zu tun hat. Der andere Grund ist jedoch, dass durch die Verwendung des Spring Boot Frameworks eine einfache H2-Datenbank Einbindung schon mitgeführt wird und somit einfach mit ins Projekt eingebaut werden kann.

### **Einbau der Datenbank in das Projekt**

Die Einbindung der Datenbank wird durch das Spring-Framework sehr vereinfacht. Die Datenbank liegt in diesem Projekt genau im entwickelten Programm. Um etwas genauer zu gehen, liegt diese H2-Datenbank in einer Text Datei mit der Bezeichnung „h2.hdb.mv.db“ in den Ressourcen des Maven Projekt. Die Datei wird beim Start des Programms aufgerufen und somit wird die Datenbank gleichzeitig gestartet. Die genaueren Aufrufe des Programms auf die Datenbank werden jedoch im Kapitel 7.2 beschrieben.

1. Schnittstellen[[61]](#footnote-61)

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, dass ein bestimmter Request immer 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.

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 4.5 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.

1. Beschreibung des Quellcodes

Die Beschreibung des Quellcodes orientiert sich an der Aufteilung der Expertengruppen und wird deshalb in zwei voneinander getrennten Unterpunkten beschrieben.

* 1. Frontend

Dieser Abschnitt beschreibt das Vorgehen und die wichtigsten Aspekte bei der Entwicklung des Frontends.

Wie in den vorherigen Kapiteln bereits erwähnt, haben wir uns im Frontend für eine Kombination aus HTML, CSS und JavaScript entschieden. Im Folgenden werden die einzelnen Dokumente näher erläutert werden.

* + 1. Entwicklungsablauf

Zunächst wurde überlegt, welche Seiten benötigt werden und was auf den jeweiligen Seiten zu sehen sein muss. Schlussendlich fiel unsere Entscheidung auf eine Login Seite und auf die fünf Unterseiten, Dashboard, Ci-Typ anlegen, Ci-Record anlegen, eine Profilseite, sowie eine Hilfeseite.

In der Abbildung x ist ein beispielhafter Entwurf der Dashboard Seite zu sehen.

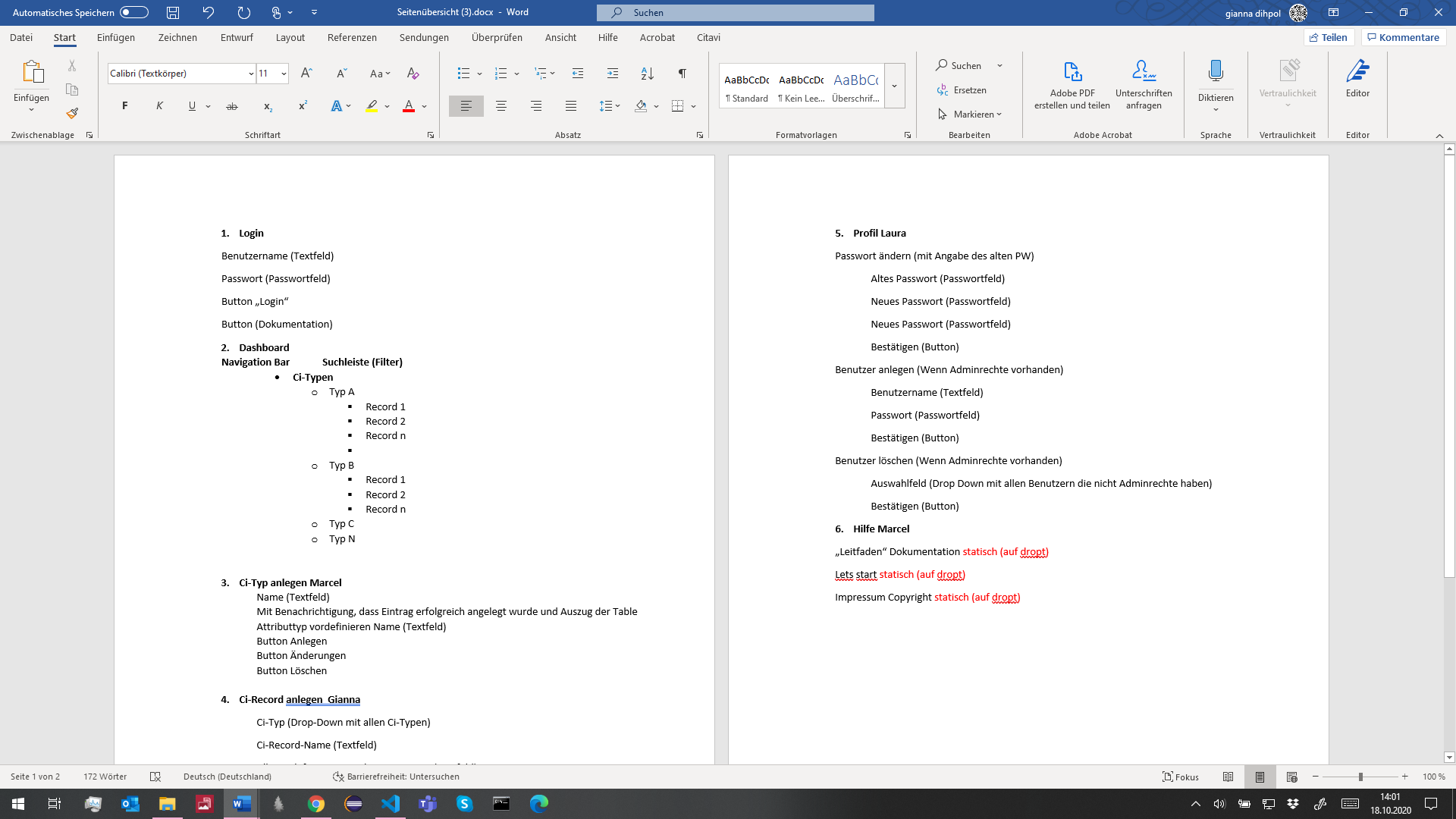


Abb. 12: Entwurf Dashboard

Nachdem für alle Seiten ein Entwurf erarbeitet wurde, haben wir uns über ein einheitliches Design der Webseite Gedanken gemacht. Dabei sind wir im Internet auf ein Tutorial gestoßen, welches uns optisch angesprochen hat.

Dies sah wie folgt aus:



Abb. 13: Tutorial zu Sidebar von CodingMarket

Dieses Design haben wir als Vorlage benutzt und zunächst die Sidebar zusammen gestaltet, da diese das Grundgerüst aller Unterseiten darstellt. Wichtig zu erwähnen ist dabei, dass wir den gesamten Code selbst geschrieben und nicht den Code des Tutorials benutzt haben, da dieser zu kompliziert gestaltet war. Schließlich sah die fertige Sidebar wie folgt aus:



Abb. 14: Sidebar

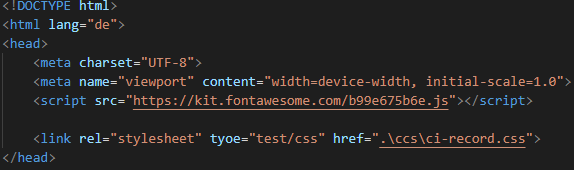
In dem darauffolgenden Schritt haben wir die einzelnen Unterseiten auf die vier Teammitglieder des Frontends verteilt. Dabei ging es erstmal nur darum schnell ein Frontend für das Backend zu Verfügung zu stellen, sodass das Backend mit Implementierungsarbeiten beginnen konnte. Danach haben wir für jede Unterseite ein separates CSS-Dokument angelegt, damit es zu keinen Konflikten kommt, wenn jedes Teammitglied gleichzeitig daran arbeitet.

Bevor jedoch jedes Teammitglied seine jeweilige Unterseite mit CSS gestaltet hat, haben wir zusammen Designentscheidungen getroffen, wie beispielsweise eine einheitliche Schriftart, Schriftgröße oder Farben.

* + 1. Beschreibung der technischen Umsetzung

Jedes HTML-Dokument ist weitgehend gleich strukturiert. Daher wird zunächst die Struktur, die sich durch alle Unterseiten zieht, erläutert und danach die Besonderheiten der einzelnen Unterseiten beschrieben.

Jedes HTML-Dokument beginnt mit diesem Ausschnitt, welcher das Grundgerüst jedes HTMLDokuments darstellt.

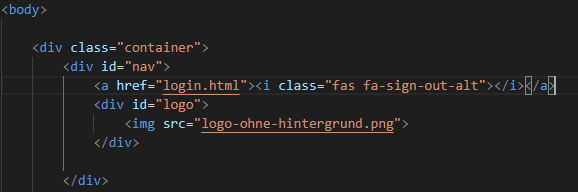


Eine Ausnahme stellt dabei der Link zu „Fontawesome“ dar, durch welchen uns ermöglicht wird lizenzfreie Icons in HTML einzubinden. Diese Einbindung erfolgt, indem man auf der Webseite fontawesome.com nach einem passenden Icon sucht und anschließend den dazugehörigen Code in das HTML-Dokument einfügt.

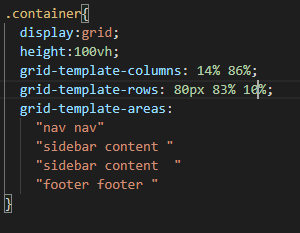
Dieser sieht dann beispielsweise wie folgt aus: <i class="fas fa-link"></i>.

Darüber hinaus beinhaltet die „Head“ - Sektion die Verlinkung des jeweiligen CSS-Dokuments.

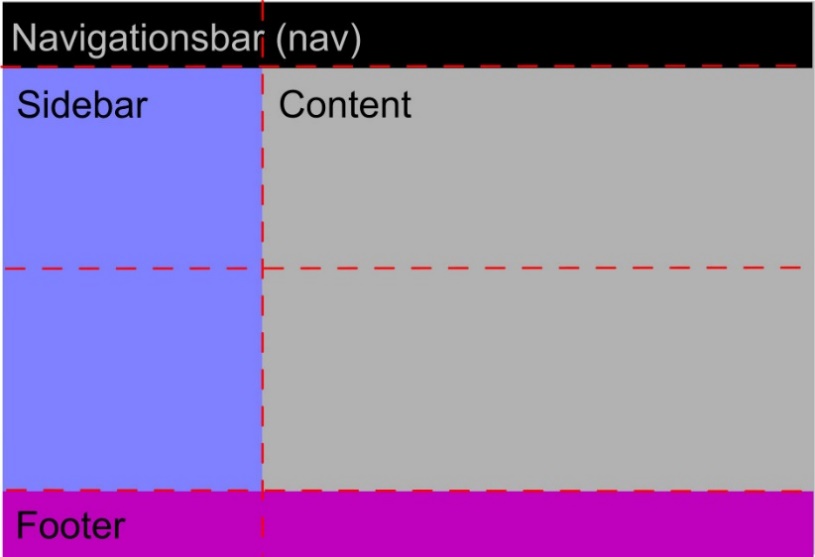
Nach dieser Sektion folgt der „Body“ des HTML-Dokuments. Diese Sektion beinhaltet zunächst einen Container, in welchem wiederum die Navigationsbar enthalten ist. Der Code sieht hierzu wie folgt aus:



Diese Container Klasse wird im CSS-Dokument durch ein Grid-Layout definiert.

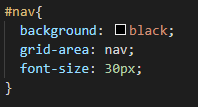


Dabei gibt es zwei Spalten und vier Reihen. Die erste Reihe besteht dabei nur aus der Navigationsbar, da diese sich horizontal über die gesamte Webseite zieht.



In diese Navigationsbar wird zunächst im HTML-Dokument das Logout Icon eingefügt, welches sich in der Navigationsbar rechts befindet. Außerdem wird auch das selbst entworfene Logo eingefügt.

 Wichtig ist hier in dem CSS-Dokument festzulegen, dass die Sektion mit der Id „nav“ auch der Grid-Area „nav“ entspricht. Dasselbe muss entsprechend auch mit „content“, „sidebar“ und „footer“ gemacht werden.



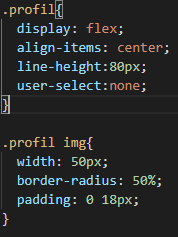
Nach dieser Navigationsbar folgt die Sidebar, welche sich auch innerhalb desselben Containers befindet.



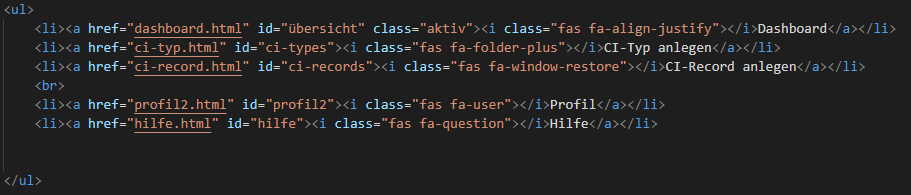
Hier wurde zunächst die Klasse „Profil“ angelegt, in welcher der Name der angemeldeten Person steht.



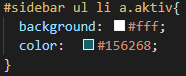
In dem CSS-Dokument wurde dieses Feld wie folgt umgesetzt. Wichtig ist hier vor allem das „user-select“, da dieses Feld nicht klickbar oder änderbar ist. Das dazugehörige Profilbild wurde mit Hilfe von „border-radius“ rund dargestellt.



Innerhalb der Sidebar folgt dann nur noch eine Liste für die Navigation zwischen den einzelnen Seiten.



Die Klasse „aktiv“ zeigt auf welcher Seite sich der User gerade befindet, indem das jeweilige Feld in der Sidebar weiß hinterlegt ist. Dies wird mit Hilfe von dem CSS-Dokument umgesetzt.



Dann folgt innerhalb jedes HTML-Dokuments die Sektion „content“. Diese unterscheidet sich bei jeder Unterseite.



Zuletzt folgt nur noch der Footer.

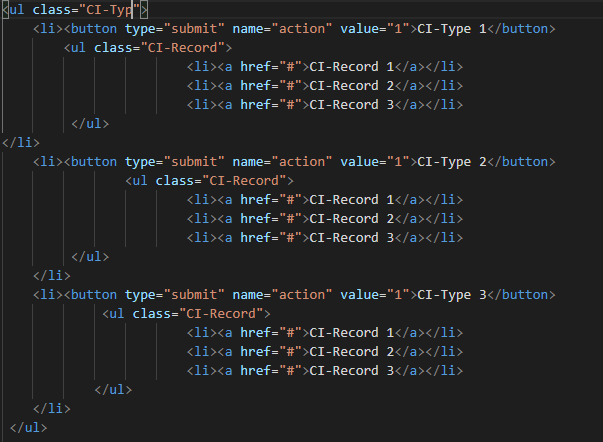
* + 1. Quellcode der Unterseiten

Im Folgenden wird die Sektion „content“ aller Unterseiten beschrieben, da diese Sektion bei allen Seiten aufgrund der unterschiedlichen Anforderungen unterschiedlich umgesetzt wurde.

Dashboard.html

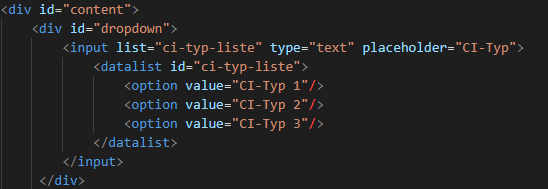
Hier wurde eine Übersicht aller Ci-Typen mit ihren dazugehörigen Ci-Records erstellt.

Dies wurde mit Hilfe mehrerer Listen im HTML-Dokument umgesetzt.



Ci-Record.html

Zunächst wird hier ein Dropdown-Menü erstellt.

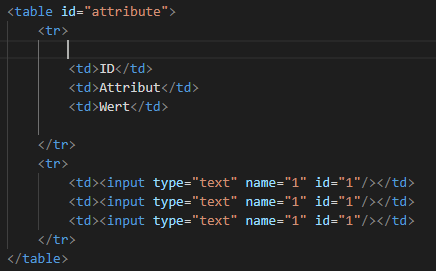


Dieses sieht wie folgt auf der Webseite aus:



In diesem Dropdown-Menü kann man dann einen der bereits vorhandenen CI-Typen auswählen und innerhalb dieses Typs ein Ci-Record anlegen, etwas an einem schon vorhandenen Ci-Record ändern oder einen Ci-Record löschen.

Als nächstes folgt eine Tabelle, in welcher die Attribute mit dem dazugehörigen Wert eines Ci-Records angezeigt werden. Hierfür wurde die Kopfzeile der Tabelle beschrieben und darunter jeweils ein leeres Textfeld. Im Backend wurde hier eine Schleife erstellt, sodass die Tabelle sich dynamisch erweitert, je nachdem wie viele Attribute es in der Datenbank gibt.



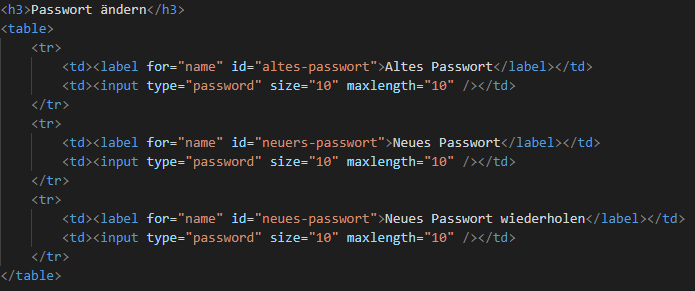
Ci-Typ.html

In diesem HTML-Dokument wurde dieselbe Tabelle, wie bei Ci-Record.html eingefügt.

Profiluser.html / Profiladmin.html

Die Unterseite Profil für den Admin beinhaltet die vier Funktionen: Benutzername ändern, Passwort ändern, Benutze anlegen, sowie Benutzer löschen.

Diese Funktionen wurden hauptsächlich mit Tabellen umgesetzt, damit die Textfelder in einer Linie untereinander angeordnet sind. Wichtig bei den Textfeldern, in welche Passwörter eingegeben werden sollen, ist den „type“ auf „password“ zu setzten. Dadurch wird das Passwort mit den dafür typischen schwarzen Punkten dargestellt.



Hilfe.html

Auf der Unterseite Hilfe können, durch den Einsatz von Links, zwei verschiedene Informationen abgerufen werden. Eine Anleitung, die sich hinter dem Link „lets start“ verbirgt und das Impressum. Unter „lets start“ findet sich das Benutzerhandbuch, welches ebenfalls in dieser Dokumentation, im Kapitel 10 Benutzerhandbuch, vorhanden ist. Das Impressum wurde mithilfe eines online-Konfigurators, auf der Seite <https://www.e-recht24.de>, erstellt. Das Impressum zeigt neben der Anschrift und den Kontaktdaten zusätzlich einige allgemeine Informationen zum Projektteam und der Website.

Login.html

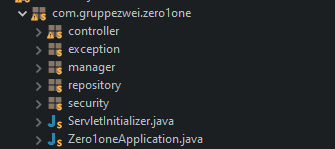
Die Seite des Logins orientiert sich an <https://codepen.io/colorlib/pen/rxddKy>. Jedoch wurde sehr viel verändert oder weggelassen.

Bei den Farben haben wir uns an dem Farbschema der restlichen Seiten orientiert und zusätzlich noch das Logo miteingebunden.

Im Unterschied zu den anderen Seiten musste hier, aufgrund eines Bugs, die CSS in den HTML-Code eingebunden werden.

* 1. Backend / Webanbindung

Im Backendteil der Anwendung der Fallstudie wird das Framework Spring verwendet. Für einen übersichtlichen Aufbau wurde die gesamte Anwendung in einzelne Teile unterteilt. Diese Unterteilung spiegelt sich in der Package-Struktur wider. Unter dem Hauptpackage com.gruppezwei.zero1one befindet sich der gesamte Java Quellcode. Direkt unter diesem Package befinden sich nur zwei Klassen. Beide erfüllen die gleiche Aufgabe, jedoch für unterschiedliche Umgebungen. Der ServletInitizializer wird für den Start auf einem Server benötigt. Die andere Klasse ist für den lokalen Start auf einem Computer zuständig.



* + 1. Auslieferung

Da Springanwendungen mit diesem Aufbau tendenziell eher auf Servern ausgeliefert werden, ist eine lokale Auslieferung nicht ganz einfach. Da Spring im inneren einen Tomcat Webserver bereitstellt, kann die Anwendung problemlos aus der Entwicklungsumgebung gestartet werden. Mit einem Plugin für den Build kann man festlegen, welche Klasse beim Aufruf zuerst angesprochen werden soll.

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <executions>  <execution>  <configuration>  <mainClass>com.gruppezwei.zero1one.Zero1oneApplication</mainClass>  </configuration>  </execution>  </executions>  </plugin>  </plugins>  </build> |

Im Fall einer lokal laufenden Springapplikation ist das die Zero1oneApplication.java. Diese wird gestartet, wenn das .war-verpackte Projekt aufgerufen wird.

Damit ein sauberer Startup und eine direkte Verlinkung auf die Startseite im Browser gewährleistet werden kann, gibt es ein kleines Hilfsprogramm. Dieses ist als Runnable Jar ausführbar und startet ein in Java implementiertes Graphic User Interface (GUI). Dieses Interface bietet zum einen Link zum Git-Repository. In diesem ist der gesamte Quellcode des Projekts einsehbar. Ein anderer Button startet das eigentliche Programm und leitet den Nutzer mit dem richtigen Startlink zum Standardbrowser. Die GUI wird daraufhin geschlossen, da das gesamte Programm im weiteren Verlauf aus dem Browser heraus bedienbar ist.

Das gesamte Auslieferungspaket besteht aus einer Anwendung zum Starten. Die Hauptkomponente ist das Springprojekt, das als .war verpackt ist. Als dritte und letzte Komponente wird die H2-Datenbank als File-Database ausgeliefert. Diese hat zum Auslieferungszeitpunkt die bestehenden Berechtigungen und einen initialen Nutzer gespeichert. Der initiale Nutzer heißt admin und hat das Passwort admin.

* + 1. Zentrales Management

Der wichtigste Teil der Anwendung ist der manager, er ist das Rückgrat des Programms. Dieses Package ist der Dreh- und Angelpunkt. Der Manager kümmert sich um die Verwaltung der Datenbankabfragen, das Mapping der Abfrageergebnisse und das Persistieren der Configuration Items. Im Inneren befinden sich fünf Manager, die jeweils auf einen Aufgabebereich spezialisiert sind. Der ReadManager liefert die Methoden, die später aufgerufen werden, um die HTML-Oberfläche mit Daten zu füllen. Die Methoden, welche die eigentlichen Abfragen ausführen stehen in Repositorys.

|  |
| --- |
| @Autowired  ConfigitemRepository confItemRepo; |

Diese werden in allen Managern mit der @Autowired- Annotation instanziiert. Wie im theoretischen Teil zu Spring erläutert wird dies verwendet, um die Lebenszyklen der Instanzen bzw. Beans voneinander zu trennen. Damit Spring diese Annotationen im Gesamtkontext erkennt, muss die gesamte Manager-Klasse mit @Component annotiert werden. Die einzelnen Methoden rufen die Methode des Repositorys auf und triggern die Query. Diese liefert eine Liste der abgefragten Objekte. Diese werden nun im Manager gemappt und so zusammengeführt, damit sie im Frontend präsentiert werden können.

|  |
| --- |
| public List<CiRecord> getCiRecordAll() {  return confItemRepo.findAll()  .stream()  .map(this::convertToCiRecord)  .collect(Collectors.toList());  } |

Das Mapping erfolgt für jedes einzelne Objekt in der Liste. Diese wird im Quellcode mit Lambda-Ausdrücken realisiert, welche seit Java 8 den Code übersichtlicher gestalten und Aufgaben wie das Mapping erheblich erleichtern. Im Wesentlichen wird für jedes Objekt der Liste, hier ConfigItems, die Methode convertToCiRecord ausgeführt. Diese liefert als Antwort ein CiRecord. Im Unterschied zu einem ConfigItem besitzt ein CiRecord noch eine Liste mit allen Attributen, die zu diesem ConfigItem gehören. Diese Liste wird ebenfalls in der Mappingmethode abgefragt und in ihren entsprechenden typ gepackt. In der Praxis wird also eine Abfrage für die Liste der ConfigItems abgesetzt und dann für jedes weitere Item noch eine Abfrage. Dies ist für einen Einsatz mit vielen Benutzern sehr unpraktisch und beeinträchtigt die Geschwindigkeit der Datenbank und somit die der Anwendung erheblich, wenn viele Nutzer gleichzeitig eine Anfrage senden. Da diese Anwendung nur lokal mit höchstens einem Benutzer eingesetzt werden soll ist jedoch kein Performancenachteil zu bemerken. Da sich zudem Joins mit Spring nur mit erhöhtem Aufwand umsetzen lassen, ist diese Vorgehensweise zu bevorzugen.

Der PersistenceManager und der UpdateManager kümmern sich um das Speichern der angelegten oder veränderten Configuration Items. Die grundlegende Funktion ist analog zum ReadManager, mit einer umgekehrten Richtung des Datenflusses. Hier werden die Objekte wieder in umgekehrter Reihenfolge gemappt und mit entsprechenden Querys abgespeichert. Der größte Unterschied der beiden Speicher-Manager ist, dass der PersistenceManager neue Objekte abspeichert und somit neue Id‘s zu den Items hinzufügen muss, damit diese im weiteren Gebrauch eindeutig identifiziert werden können.

Der DeleteManager leitet die Löschanfragen. An dieser Stelle werden keine Objekte abgefragt oder umgewandelt. Die einzelnen Items werden über die ID gelöscht. Der AuthenticationManager kümmert sich um die Abfrage und Verwaltung der Benutzerdaten. Es gibt also in der gesamten Anwendung kein Vorgang, der nicht durch den Manager gesteuert wird.

* + 1. Datenbankzugriff

Der Wurzel der Datenhaltung bilden die Repositories im repository-Package. Neben den Repositories befinden sich hier auch die POJOs, die zur Abfrage benötigt werden. Diese Plain Old Java Objects (POJO) haben die Aufgabe den Inhalt einer Datenbankabfrage wieder zu geben. Im Regelfall bilden die POJOs eine Zeile der Tabelle ab, die abgefragt wird. Sie stellen also eine Entität dar, die in dieser Tabelle gespeichert ist. Aus diesem Grund werden diese Klassen auch mit @Entity annotiert.

|  |
| --- |
| @Entity  public class Configitem {  @Id  private int id;  private String configitemtypname;  private String name;  //Getter and Setters  } |

Durch diese Annotation weiß Spring, dass diese Klasse eine Entität auf der Datenbank wiederspiegelt. Damit eine saubere Abfrage erstellt werden kann, muss der Schlüssel auf der Datenbanktabelle mit @Id gekennzeichnet werden. Die Typen der Attribute müssen hier nicht zwangsläufig mit denen der Datenbank übereinstimmen. Eine Zahl oder ein Datum können auch als String abgespeichert werden. Umgekehrt funktioniert dies in Java nicht. Der Name der Attribute sollte hingegen mit den Spaltennamen der Datenbank übereinstimmen. Dies gilt auch für den Namen der Klasse. Falls diese nicht übereinstimmen, oder man sicher gehen möchte, dass die richtige Tabelle und die richtigen Spalten angesprochen werden, kann man auch Namen zuordnen. Auch hier arbeitet Spring mit Annotationen. Man kann bereits hinter @Entity den Namen spezifizieren oder nutzt die eigene Annotation @Table:

|  |
| --- |
| @Table(name=”Spaltenname”) |

Analog gibt es auch eine Annotation @Column. Speziell für die Tabelle kann man auch den Namen des Schemas angeben. Stimmen die Namen aber mit den Klassen- und Attributnamen überein, besteht kein Bedarf für diese Annotationen. Nur @Entity und @Id sind zwingend erforderlich.

|  |
| --- |
| @Repository  public interface ConfigitemRepository   extends JpaRepository<Configitem, Integer> {    List<Configitem> findByName(String name);    List<Configitem> findTopByOrderByIdDesc();  } |

Ein Repository übernimmt die eigentliche Aufgabe der Abfrage. Im Gegensatz zu den anderen Teilen wird ein Repository als Interface implementiert. In dieser Anwendung erben alle Repositories von einem JpaRepository. Dieses erbt ursprünglich von einem CRUD-Repository. Wie im Namen CRUD schon zu erkennen ist, ermöglicht dieses Repository die grundlegenden Operationen Create, Read, Update und Delete. Das Repository ist generisch aufgebaut und benötigt zwei Parameter: Die Entity, die abgefragt werden soll, und den Datentyp des Schlüssels. Im obigen Beispiel sind dies die Entity Configitem und der Primärschlüssel vom Typ Integer. Mit diesen Informationen kann Spring bereits Datenbankabfragen durchführen. Voraussetzung ist, dass in der application.properties die Zugangsdaten und der Treiber der Datenbank hinterlegt sind.

|  |
| --- |
| spring.datasource.url=jdbc:h2:file:./src/main/resources/data/h2.hdb  spring.datasource.driverClassName=org.h2.Driver  spring.datasource.username=user  spring.datasource.password=password |

Wie im Manager bereits erwähnt wird ein Repository immer autowired. Es wird als vom Spring-Anwendungskontext als Bean zur Verfügung gestellt. An dieser Stelle wird auch deutlich welche große Aufgabe das Framework übernimmt. Es baut mit den Informationen über die Datenbank eine Connection auf. Diese muss nicht manuell erzeugt werden. Dadurch entfällt das Aufbauen der Verbindung vor der Abfrage und das Abbauen danach. Durch diese Verwaltungsarbeit des Frameworks bleibt hier der Quellcode übersichtlich. Außerdem entfällt eine Implementierung des Interfaces, auch diese Aufgabe wird von Spring übernommen.

Das Repository beherrscht bereits einfache Abfragen wie findAll() oder findById(<S> id). Die Methoden antworten entweder mit einer Liste von Configitems, oder mit einem Optional-Objekt, je nachdem wie viele Ergebnissätze erwartet werden. Zudem sind auch Methoden delete() zum Löschen und zum Speichern save() vorgesehen. Ein Objekt kann entweder über die Id oder über die Entität selbst gelöscht werden. Die Speichermethoden können sowohl zum Neuanlegen als auch zum Aktualisieren verwendet werden. Gibt man ein Objekt mit, dessen Primärschlüssel schon existiert, werden alle restlichen Parameter überschrieben.

Sind jedoch kompliziertere Abfragen nötig, um gewünschte Ergebnisse zu erhalten, hat man zwei Möglichkeiten. Erstens kann man eine Methode im Repository mit @Query versehen und schreibt dahinter in Klammern eine eigene Query. Diese muss als Java Database Connectivity (JDBC)-konforme Query geschrieben werden, damit Spring diese, mit dem in der pom.xml hinterlegten Treiber, zu einer ausführbaren Abfrage umwandeln kann.

Als zweite Möglichkeit kann man Methoden vorgeben, die bestimmte Schlüsselwörter im Namen tragen. Ein simples Beispiel ist oben im ConfigitemRepository gegeben. Nennt man die Methode findBy und anschließend das gewünschte Attribut, nach dem gesucht werden soll, dann bildet Spring eine Query, die genau dies macht. Diese Schlagworte können wesentlich komplizierter werden. Im Beispiel darunter wird die oberste Entität zurückgegeben, nachdem die Tabelle absteigend nach ihrer ID geordnet wurde. Es wird also das Element mit der größten ID zurückgegeben.

Um an die Attribute zu kommen, die zu einem CiRecord gehören, muss man normalerweise einen Join durchführen. Da wir aber in einem Repository immer nur exakt eine Tabelle wiederspiegeln, ist dies nicht so einfach. Es kann natürlich ein Repository erstellt werden, das ein solches Ergebnis entgegennimmt. Dies ist aufwändiger als mehrere Abfragen durchzuführen. Da wie bereits erwähnt die Geschwindigkeit der Abfragen bei dieser Single-User-Anwendung eine untergeordnete Rolle spielt, wurde auf die zweite einfachere Variante zurückgegriffen.

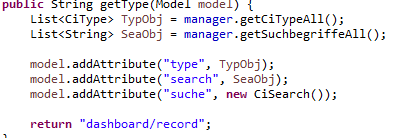
* + 1. Dateien, Struktur und Übermittlung

Um die Webanbindung der Applikation zu ermöglichen, mussten zuerst ein Controller und ein HTML-Template dem Projekt hinzugefügt werden. Zum Testen während der Entwicklung wurde auf den Spring internen Apache Tomcat Server zurückgegriffen. Hierfür mussten keine weiteren Voreinstellungen vorgenommen werden. Der Tocmcat-Server startet automatisch auf Port 8080.

Um nun alles für die Darstellung der Software vorzubereiten, haben wir ein Controller-Package namens „com.gruppezwei.zero1one.controller“ im Java-Projekt angelegt. Darin wurden die Controller-Klassen, wie bereits im Kapitel über Thymleaf beschrieben, angelegt. Begonnen mit dem „Homecontroller“, der für das Dashboard und somit für das Anzeigen der Ci-Typen und Ci-Records zuständig ist. Innerhalb des Controllers wurden „@SpringBootApplication“ zur Ausführbarkeit der Klasse und „@Controller“ um Spring zu sagen, dass es sich um einen Controller handelt. Außerdem wurde mit „@Autowired“ eine Verbindung zur Java-Klasse „PersistenceManager“ geschaffen, um auf die Methoden zum Abrufen und Speichern in der Datenbank zuzugreifen.

Innerhalb der „public class Homecontroller“ wurden dann die Get- und Post-Methoden mit „@GetMapping“ oder „@PostMapping“ definiert. In Klammern dahinter wird als „value“ die URL, unter der die nachfolgende Java-Methode erreichbar ist, mitgegeben. Beispielsweise: @GetMapping(value = "/dashboard"

Danach folgt die Methode, die ein Model mitgegeben bekommt. In diesem Model können beliebig Attribute gespeichert werden, die dann an das Frontend übermittelt werden. Innerhalb der Methode werden dann alle Daten Java-Seitig vorbereitet und an das Model übergeben.



Beispielsweise wird in dieser Methode zuerst eine Liste von CiType erstellt. Diese bekommt durch die Methode „getCiTypeAll()“ aus dem „PersistenceManager“ alle CiTypen geliefert. Diese Liste wird nachher zur Darstellung aller vorhandenen Ci-Typen benötigt. Danach wird eine Liste vom Datentyp String erstellt, die von der Methode „getSuchbegriffeAll()“, die ebenfalls im „PersistanceManager“ definiert ist, alle Suchbegriffe bekommt. Diese Suchbegriffe sind dann in der Suche als Vorschlag sichtbar.

Beiden Listen werden dann in den Zeilen darunter an das Model übergeben. Im Model sind diese dann unter „type“, bzw. „search“ aufrufbar. Außerdem geben wir dem Model noch ein leeres Objekt vom Datentyp CiSearch mit. In diesem wird nachher unser Suchwort gespeichert und dann wieder über eine Post-Methode an die Java-Applikation zurückgegeben.

Im Return der Methode wird ein String zurückgegeben. Dieser beinhaltet die Adresse des Templates, das für die Darstellung genutzt werden soll. In diesem Fall heißt das Template „record“ und befindet sich im Ordner „dashboard“.

Wie bereits im Kapitel Thymleaf beschrieben, sind die Templates HTML-Dateien, die im Templates-Ordner in den „resources“ des Java-Projekts abgespeichert sind. Diese Templates wurden von den Kollegen. die das Frontend gestaltet haben, erstellt.

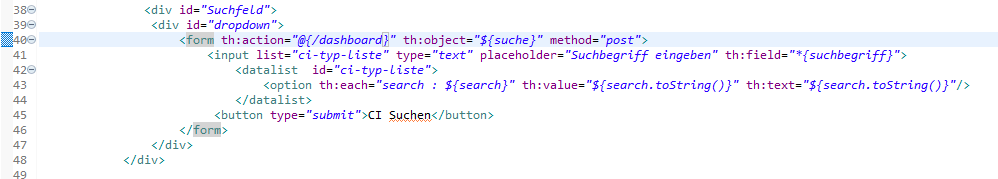
* + 1. Darstellung auf dem Client

Wie die eben übermittelten Daten nun dargestellt werden, ist mittels bestimmter Thymleaf-Syntax beschrieben.



Hier sieht man den typischen Head einer HTML-Datei. Zuerst wurde dem Template die Ergänzungen in Zeile 2 „xmlns:th=*"http://www.thymleaf.org/"*> hinzugefügt, um die Datei für Thymleaf vorzubereiten. Außerdem wurde in Zeile 6 der Quellordner für Inhalte wie Bilder oder CSS-Dateien angegeben, um die Daten, die bereits über HTML eingebunden waren, entsprechend zu hinterlegen.

Schauen wir uns nun exemplarisch die Einbindung der Suche an.



In der Zeile 40 wurde ein <form> Tag, der bis Zeile 46 geht, hinzugefügt. Dazwischen haben wir alle wichtigen Funktionen für die Suche untergebracht. Der Form-Tag ist ein HTML-Tag und kennzeichnet ein Formular. Im ersten Tag sehen wir bereits typische Thymleaf-operationen. Die „th:action“ gibt dabei an, an welche Post-Methode die Eingabe gesendet werden soll. „th:object“ betitelt das Obejekt, das übermittelt werden soll. In unserem Fall ist es das „suchen“ Objekt, das wir unserem Model mitgegeben haben. Dies ist vom Datentyp „CiSearch“. Außerdem definieren mit „method“ noch die Übermittlungsmethode. Dies ist eine Standard http-Methode.

Mit dem nachfolgenden „<input>“ Tag erstellen wir ein Feld, in das der Benutzer etwas eintragen kann. In unserem Fall das Suchfeld. Hier benutzen wir „th:field“\*{Suchbegriff}“>“, um den eingegebenen Begriff in den String „Suchbegriff“§ innerhalb des CiSearch-Obejkts zu speichern. Dieser String ist in der entsprechenden Java-Klasse definiert. Die <datalist> ermöglicht, dass eine Liste mit Vorschlägen zur Verfügung steht.  
Sie wird mit <option th:each=*"search : ${search}"* th:value=*"${search.toString()}"* th:text=*"${search.toString()}"*/> befüllt. Dabei wird mit “th:each” jedes Objekt der Liste, die im Model unter „search“ gespeichert ist, angefasst und mit „th:text“ als „option“ in die Aufklappliste aufgenommen.

In Zeile 45 ist Button vom Typ Submit definiert. Dieser löst die ganze Aktion aus und veranlasst somit das Senden des Suchbegriffs über eine Post-Methode.

* + 1. Sicherheit

Das Sicherheitskonzept und die Benutzerverwaltung sind mit Spring Security umgesetzt. Dieser sichert alle Seiten ab, die von unserer Anwendung verwaltet werden. Im eigenen Package, der security, wird die SecurityConfig implementiert, welcher die zentralen Sicherheitsfunktionen verwaltet. Sie erbt vom WebSecurityConfigurerAdapter. Zum einen wird hier die Methode configure überschrieben, die dafür zuständig ist den Nutzer zu authentifizieren. Dazu ruft sie den CustomAuthenticationprovider auf. In der anderen configure-Methode wird die HttpSecurity konfiguriert.

|  |
| --- |
| @Override  protected void configure(HttpSecurity http) throws Exception{  http.authorizeRequests().antMatchers("/resources/\*\*")  .permitAll().anyRequest().authenticated().and()  .formLogin().loginPage("/login")  .defaultSuccessUrl("/dashboard",true)  .permitAll().and()  .logout().permitAll();  } |

Hier werden die essenziellen Verhaltensweisen der Sicherheit festgelegt. Als erstes wird fesgelegt, dass alle Requests akzeptiert werden. Mit .authenticated() wird konfiguriert, dass jeder Request authentifiziert sein muss, sonst wird er nicht bearbeitet. Mit .formLogin() wird ein eigenes Loginformular registriert, welches unter der Url /login aufgerufen wird. Damit dieses aufgerufen werden kann wird die Url im SecurityController gemappt und festgelegt, welches Loginformular aufgerufen werden soll, wenn eine Getanfrage eingeht. Mit .defaultSuccessUrl(..) wird die url hinterlegt, die nach dem Login aufgerufen werden soll.

Im CustomAuthenticationprovider wird der standard AuthenticationProvider überschrieben. An dieser Stelle wird durch den AuthenticationVerifier der AuthenticationManager aufgerufen. Dieser fragt den Nutzer und das Passwort ab. Das Passwort selbst wird nur als Hashwert gespeichert und mit dem verhashten Wert der Eingabe abgeglichen. Der AuthenticationVerifier gibt nach erfolgreicher Anmeldung einen AuthenticationToken zurück, der als Cookie gespeichert wird. Über diesen authentifiziert sich der Browser nun auf allen Seiten. Die Eingabe des Passworts ist clientseitig nicht komplett geschützt. Die Eingabe wird erst auf dem Server verhasht. Diese ist für eine Online-Anwendung sehr schlecht. Aber aufgrund des Anwendungsfelds des Programms ist dies keine große Sicherheitslücke, da die Anwendung lokal betrieben wird.

* + 1. Fehlerhandling

Um das Programm der Fallstudie abzurunden muss noch ein geeignetes Fehlerhandling implementiert werden. Auch hier bietet Spring einen Default an, der optisch jedoch wenig ansprechend ist. Für eine eigene Fehlerseite wird für clientseitige Fehler die Standardposition für eigene Fehlerseiten benutzt. Legt man unter src/main/ressources/templates eine HTML-Seite mit dem Namen error.html ab, ruft Spring diese automatisch im Fehlerfall auf. Durch einfache Thymeleaf-Befehle werden auf dieser der Fehlerstatus und die textuelle Beschreibung des Fehlercodes ausgegeben. Da im Fall eines internen Fehlers so nur eine unschöne 500 Internal Server Error Meldung ausgegeben wird, werden diese Fehler auf eine andere Weise gehandled.

Damit eine individuelle Fehlermeldung für interne Fehler ausgegeben werden kann, wird im Package exception ein ExceptionHandler implementiert. Dieser wird als @ControllerAdvice annotiert. Damit wird sichergestellt, dass Spring die hier verwalteten Fehler überprüft, bevor die Standardmeldung gesetzt wird.

|  |
| --- |
| @ExceptionHandler(EmptyResultDataAccessException.class)  public String handleException(EmptyResultDataAccessException ex){  return "errormanual"  } |

Das Beispiel zeigt eine vereinfachte Handler-Methode. Diese ExceptionHandler reagiert auf eine EmptyResultDataAccessException. Wird dieses Exception irgendwo im Programm geworfen, schaut Thymeleaf nach einer errormanual.html im templates-Ordner und gibt diese HTML-Seite aus. Die weiteren Methoden wurden wie in den Controller-Klassen um ein Model erweitert, welches eine passende Fehlermeldung mit ausgeben kann.

Im gleichen Package sind auch alle eigenen Exceptions abgelegt. Durch die Erstellung eigener Exceptions kann die Ausgabe im Fehlerfall sehr genau ausfallen, da jeder Fehlerfall seine eigene Fehlermeldung hat. So kann gewährleistet werden, dass der Nutzer weiß, was nicht funktioniert, ohne diesem eine Auskunft über den internen Aufbau der Anwendung zu geben. Dies wäre besonders relevant, wenn die Anwendung der Fallstudie auf einem Webserver für externe Nutzer zugänglich wäre.

* 1. Codequalität

Mit Codequalität wird in der [Programmierung](https://de.wikipedia.org/wiki/Programmierung) das Maß bezeichnet, mit dem der [Quellcode](https://de.wikipedia.org/wiki/Quelltext) eines [Programms](https://de.wikipedia.org/wiki/Computerprogramm) bestehende, die formale Gestaltung betreffende Anforderungen erfüllt. Die Codequalität ist dabei eine nicht-funktionale Anforderung, die zur Verständlichkeit, Analysierbarkeit und Modifizierbarkeit beitragen soll. Hier soll vor allem auf die Modifizierbarkeit eingegangen werden.

Eine Möglichkeit dieses Zeil zu erreichen ist die Aufstellung von Programmrichtlinien. Dadurch kann man den Code einheitlicher gestalten, auch wenn mehrere Entwickler gleichzeitig daran arbeiten. Im Projekt wurde dies vor allem durch eine einheitliche CSS Struktur umgesetzt, die auf alle Unterseiten importiert wurde. Auch im HTML-Code wurde versucht einen einheitlichen Programmierstil zu integrieren. Je einheitlicher ein Programm geschrieben ist, desto leichter ist nicht nur die Wartbarkeit, sondern auch die Erweiterbarkeit.

Eine weitere Möglichkeit ist es, für eine niedrige Komplexität zu sorgen. Dabei kommt es darauf an, auf so viele unnötige Strukturen wie möglich zu verzichten. Außerdem sollte auf eine hohe Wiederverwendbarkeit gesetzt werden. Das bedeutet, hat man einmal eine funktionierende Struktur entworfen, sollte diese möglichst oft eingesetzt werden. Dies verhindert eine unnötig hohe Komplexität, da man nicht nur einen Wiedererkennungswert generiert, sondern auch Zeitersparnis bei der Analyse des Codes. Der Entwickler selbst hat den Vorteil, dass ihm ein Menge Aufwand erspart bleibt, da er sich auf wenige funktionierende Strukturen verlassen kann. Allerdings sollte darauf geachtet werden, dass eine hohe Wiederverwendung auch eine komplexe Vererbungshierarchie mit sich bringen kann. Es sollte immer darauf geachtet werden, ob es sich eher lohnt die Wiederverwendung oder die geringe Vererbungshierarchie in den Vordergrund zu stellen. Auch darauf wurde im Projekt geachtet. Wie oben angesprochen wurde auf eine hohe Wiederverwendung auf den Unterseiten der Website geachtet. Auch im HTML-Code wurde auf eine einheitliche Metrik geachtet.

1. Qualitätssicherung

Essentiell für den Erfolg eines Software-Projektes ist die Qualität der jeweilig entwickelten Software.[[62]](#footnote-62) Ausschlaggebend ist jedoch nicht diese zu einem gewissen Zeitpunkt zu erreichen, wie beispielsweise vor der letztendlichen Auslieferung an den jeweiligen Kunden. Sondern dafür die Qualität von Beginn des Entwicklungsprozesses an, während der Entwicklungsphase, bis hin zur Fertigstellung der Software und der darauffolgenden Auslieferung zu erhalten. Anwendung findet hier eine vom ganzen Team genutzte und durchgeplante Qualitätssicherung. Folgende übernimmt das Auffinden von Fehlern und schafft folglich ein grundlegendes Vertrauen in das erarbeitete Produkt.

Vermehrt werden hier in der Softwareentwicklung umfangreiche Softwaretests benutzt, jedoch liegt das Hauptaugenmerk nach Absprache des Teams daran schon keine Fehler in die Software zu bringen. Um dies zu erreichen, werden im gesamten Projekt neben dynamischen und statischen Tests zum Finden von Softwarefehlern und zur Qualitätskontrolle des Produktes, eine ganzheitliche Qualitätssicherung verwirklicht. Dazu werden im Team organisatorische und konstruktive Qualitätssicherungsmaßnahmen eingebunden.

Unter den eingeführten konstruktiven Qualitätsscherungsmaßnahmen wird sich im Projekt eine Verhinderung und auch Minimierung der Fehlereinbringung in der Software versprochen. Hierfür gibt es die Möglichkeiten bereits erprobte Module und Architekturen weiterzuverwenden, die Einbindung eines Frameworks für die Entwicklung oder auch darin die Anforderungen an das Produkt durch Prototypen und Tests dieser zu überprüfen. Das Team hat sich dabei entschlossen sich auf die Nutzung des Spring Frameworks zur Entwicklung zu fokussieren. Dafür sprachen zum einen vorerst qualitätssicherungsunabhängige Aspekte, wie eine vereinfachte Vermittlung zwischen Frontend und dem Backend, Zugriff auf die Datenbank, sowie die vereinfachte Login Erstellung. Jedoch verhindert die Einbindung eines solchen Frameworks von vorn herein schwerwiegende Fehler am Anfang der Entwicklung, da genau die vorgefertigten und verfügbaren Methoden in das Projekt mit eingebaut werden.

Hingegen sorgen Organisatorische Qualitätssicherungsmaßnahmen für eine Grundlage der gesamten Qualitätssicherung im gesamten Team, da die Ablauf- und Aufbauorganisation eines gemeinsam erarbeiteten Projektes großen Einfluss auf die gelieferte Qualität hat. Aus diesem Grund wird diese zum wichtigsten Punkt der Qualitätssicherung im Projekt, um eine solide Basis für den Erfolg dieses Projektes zu sichern. Erfolgsabhängig für eine erfolgreiche organisatorische Qualitätssicherung stellt hierbei die verwendete Projektmanagementmethode und das genutzte Vorgehensmodell zur Qualitätssicherung. Dabei stand zum einen zur Auswahl eine klassische Software Projektmanagementmethode und das V-Modell als allgemeines Vorgehensmodell zu nutzen. Hierfür wurde im Team die Methode Scrum gewählt, da dieses sich als Rahmen einer ganzheitlichen Qualitätssicherung am besten eignet und vor allem Scrum durch dessen Ablauf eine bereits einfache Qualitätssicherung besitzt. Zunächst ist Scrum ein einfaches Prozessrahmenwerk zum Projektmanagement, worin erlaubt ist verschiedene Techniken, Methoden und Prozesse zur Entwicklung zu integrieren. Aber trotz dieser hohen Freiheitsgrade besitzt Scrum schon eine Unmenge an im Projekt hilfreichen Qualitätssicherungsmaßnahmen. Genauer gibt es in erster Linie die Ablauf- und Aufbauorganisation des bearbeiteten Projektes vor. Somit entsteht durch die Anwendung des Scrum Ablaufes eine allgemeine und einfache Qualitätssicherung und Qualitätskontrolle. Daher wurde im Team ganzheitlich entschieden die Qualität der Software nur unter der Anwendung der Projektmanagementmethode Scrum zu sichern.

Um die einfache Qualitätssicherung in dieser agilen Projektmanagementmethode zu erläutern, wird ein Blick in den Ablauf und Teile eines Scrum-Projektes geworfen. Darin stellt das gesamte Team schon einmal einen Großteil der Qualität des Projektes dar, weil durch dieses selbstorganisierte Team die Anforderungen erst realisiert werden können. Somit ist die Zusammenstellung und die Aufteilung einer der kritischen Erfolgsfaktoren für das gesamte Projekt, weshalb das Projekt in Teams für Backend, Frontend und das Mapping spezifisch derer Fähigkeiten eingeteilt wurde. In Hinblick auf die Qualitätssicherung können hier im Team zwei Maßnahmen unterschieden werden. Dass der Produt Owner jederzeit im Laufe der Projektentwicklung voll entscheidungsfähig ist und zudem, dass die Teammitglieder neben den Fähigkeiten zur Entwicklung qualitativ hochwertiger Software, auch Kenntnisse in den Methoden der Qualitätssicherung besitzen. Das ist wichtig, da in Scrum eine bestimmte Person bzw. ein dedizierter Tester in Scrum nicht vorhanden ist. Dies wurde über das gesamte Projekt hinweg durchgezogen und schon vor des Entwicklungsstarts dem Team grundlegende Qualitätssicherungsmaßnamen erläutert.

Während der Projektinitiierung übernahm das Team die Aufgabe die Rahmenbedingungen für die weitere Entwicklung des Projektes festzulegen. Dabei wurden im Sinne der Qualitätssicherung Vorgaben hinsichtlich von Programmierrichtlinien oder andern festgelegt, damit eine einfache Wartbarkeit der Software zu gewährleisten war. Im Team ist beispielsweise eine solche Richtlinie, dass keine Software die nicht zu 100% läuft und getestet wurde, in das teaminterne Git geladen wird. All diese Vorgaben des Teams zusammengefasst, bilden die Definition of Done, welches die Kriterien für die fertige Software stellt. Im Team wurde diese Defintion of Done in einer Exel Tabelle einfach gehalten.

Für die Qualitätssicherung stellte auch die Sprintplanung eine sehr große Rolle. Dadurch wurden im gesamten Team die zu implementierenden Anforderungen einem Review unterzogen, um sicherzustellen, dass nur die in der Sprintplanung geplanten und verstanden Anforderungen während des Sprints ausgeführt wurden. Somit wurden klar im gesamten Team die Rollen und die jeweiligen Aufgaben verteilt, um eine qualitativ hochwertige Weiterentwicklung überhaupt zu ermöglichen.

Diese Sprints wurden wöchentlich in einem Meeting teamintern geplant und folglich ausgeführt. Dabei zählten zu den Qualitätssicherungsmaßnahmen während dieses Sprints die zum einen statische Analyse des Codes und Testen des Codes. Sowie die Anwendung des aus Extreme Programming bekannte Pair Programming. Dabei nahm eine Person aus dem Team die Programmierarbeit vor und ein zweiter überprüfte den geschriebenen Code per Sichtprüfung. Somit konnte eine klare Qualitätssicherung während des jeweiligen Sprints erfolgen.

Nach jedem Sprint fand an jedem Donnerstag in der Woche das teaminterne Teamreview statt. Dabei wurde das im Sprint erarbeitet Produktinkrement vorgestellt und anhand der Anforderungen und der Definition of Done vom jeweiligen Produkt Owner bewertet. Dadurch konnte gewährleistet werden, dass das richtige Produkt entwickelt wird und der nächste Sprint auf einer richtigen Basis stattfindet. Teil eines jeden Sprintreviews ist jedoch auch die weitere Sprintplanung und die Rollenversteilung auf die jeweilig anfallenden Aufgaben gewesen. Wodurch eine teaminterne und qualitativ hochwertige weiterführende Entwicklung stattfinden konnte.

Um dafür zu sorgen, dass nicht nur die Qualität des Projektes und der Arbeitsweise im Team erhalten und gesichert wurde, hat das Team sich einer Sprint-Retrospektive unterzogen. Dabei wurde zunächst wie in jedem Sprint-Review, das entwickelte Produktinkrement und die erreichten Fortschritte gezeigt, jedoch diese Punkte und vor allem auch die Abläufe der im Sprint vollzogenen Entwicklung beobachtet und analysiert. Dadurch konnte im ganzen Team die Arbeitsweise verbessert werden und dadurch die allgemeine Projektqualität erhöht werden. Diese wurde aber lediglich klein gehalten und gemeinsam darüber gesprochen, was verbesserbar wäre. Und nicht genaue Ablaufs Analysen mit Unmengen an ausgearbeiteten Daten erstellt.

Um zuletzt vor der Auslieferung die endgültige Funktion und somit auch die endgültige Vollständigkeit der Software zu gewährleisten, wurde die mit Scrum durchgeführte Qualitätsicherung noch einmal um verschiedene statische und dynamische Softwaretest erweitert. Damit das fertige Projekt voll und ganz funktionsfertig ist. Durch die Anwendung dieses Scrum Ablaufes war es möglich die Qualität der entwickelten Software von Beginn der Projektinitiierung bis hin zu der Auslieferung an den Kunden zu erhalten. Da diese Projektmanagementmethode eine schon sehr genaue Qualitätssicherung hat, wurden keine weiteren qualitätssichernden Maßnahmen geplant oder verwendet.

1. Berechtigungsverwaltung

Der Abschnitt Berechtigungsverwaltung liefert im ersten Unterpunkt generelle Informationen zu diesem Thema. Der zweite Unterpunkt bezieht sich auf die generellen Informationen und spezifiziert diese in Bezug auf das durchgeführte Projekt.

* 1. Berechtigungsverwaltung generell

Der Sinn einer Berechtigungsverwaltung liegt, wie bei jeder Anwendung auf der Hand. Die Berechtigungsverwaltung ermöglicht es in Anwendungen, die Access Control Berechtigungen auf die verschiedenen Benutzer zu verteilen. Hierbei werden die Benutzer in verschiedene Benutzergruppen eingeteilt und es wird sichergestellt, dass jeder Nutzer nur diejenigen Funktionen der Anwendung nutzen kann, die er zur Ausübung seiner – für ihn vorbestimmten – Tätigkeit in der Anwendung benötigt. Außerdem wird mit der Berechtigungsverwaltung entschieden, welche Elemente ein Benutzer überhaupt in der Anwendung sehen kann und welche für ihn zur Benutzung freistehen.

Mithilfe der Berechtigungsverwaltung können wiederverwendbare Richtlinien mit bestimmten Regelsätzen erstellt werden, die den Zugriff auf bestimmte Dateien, Ordner, Datei-Hashes und einige Systemsteuerungskomponenten der Anwendung festlegen und für manche Benutzergruppen erweitern oder einschränken. Eine Berechtigungsverwaltung läuft meistens nach dem Prinzip der geringsten Rechte, also es werden Benutzern nur so viele Reiche eingeräumt, wie sie zur Ausübung ihrer Tätigkeit in der Anwendung benötigen, ohne diesen gleich Administratorrechte vergeben zu müssen

* 1. Berechtigungsverwaltung im Programm

Die Berechtigungsverwaltung in unserem Programm verläuft wie im Unterkapitel „Datenbanken“ (siehe Kapitel 5.1.5) so, dass in der Datenbank zwei Tabellen erstellt wurden, einmal die Tabelle „Berechtigungen“ und die Tabelle „User“. Die Berechtigungen der Benutzergruppen sind festgelegt und es gibt die zwei Gruppen „Administrator“ und „Benutzer“.

Bei der Erstellung eines neuen Users schreibt das Programm in die User-Tabelle den jeweiligen Namen, das Passwort in Form eines Hash und die Referenz auf die Berechtigungs-ID der Person als Fremdschlüssel. Dabei kann jedoch jeder User nur eine einzige Berechtigung besitzen und dadurch auch nur die mit der Berechtigung verbundenen Tätigkeiten im Programm ausführen. Ihm wird also eine der Benutzergruppen zugewiesen und somit entschieden, ob der Nutzer die Rechte eines Administrators nutzen kann oder nur über die Grundfunktionen verfügt. Über welche Berechtigungsgruppe der Benutzer verfügt, entscheidet der Administrator beim Anlegen des Benutzers, da Benutzer erstmal von einem Administrator mit einem vorbestimmten Passwort angelegt werden, das nach der ersten Anmeldung umgeändert werden kann.

Ein normaler Benutzer verfügt hierbei über die Grundfunktionen des Programms, er kann sich also mit seinen zuvor umgeänderten Anmeldedaten anmelden, er kann CI-Typen und dann darauf basierend CI-Records anlegen und diese auch bearbeiten. Er kann sein eigenes Profil anklicken, sich abmelden und generell die Funktionen der Anwendung nutzen. Ein Administrator hingegen verfügt über alle Grundfunktionen, kann aber auch Änderungen an der Seite vornehmen und hat so gesehen die Rechte alles zu tun wie z.B. CI-Records, die von anderen Nutzern angelegt wurden, wieder zu löschen. Er kann auch wie vorher angesprochen, neue Benutzer erstellen und ihr vorrübergehendes Passwort festlegen.

Unsere Berechtigungsverwaltung ist somit relativ simpel aufgebaut und unterscheidet nur zwischen den Administratoren und den normalen Nutzern, einfach nur, um normalen Nutzern nicht die Rechte zu geben, dass sie einfach unser Programm verändern könnten und somit der Software schaden könnten. Es dient also dazu, unser Programm vor Leuten zu schützen, die nicht in der Lage sein sollten etwas daran zu ändern.

1. Benutzerhandbuch

**Erklärung**

Wir versichern hiermit, dass wir unsere Dokumentation der Fallstudie mit dem Projekt: zero1one selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Literaturverzeichnis

**Crispin, L./Gregory, J. (2009):** Agile testing, A practical guide for testers and agile teams, Upper Saddle River, N.J: Addison-Wesley

**Dalton, J. (2019):** Great Big Agile, An OS for Agile Leaders, Berkeley, CA: Apress

**Foegen, M./Kaczmarek, C. (2016):** Organisation in einer digitalen Zeit, Ein Buch für die Gestaltung von reaktionsfähigen und schlanken Organisationen mit Hilfe von Scaled Agile & Lean Mustern, 3. Auflage, Darmstadt: wibas GmbH

**Fowler, F. M. (2019):** Navigating Hybrid Scrum Environments, Understanding the Essentials, Avoiding the Pitfalls, Berkeley, CA: Apress

**Goll, J. (2011):** Methoden und Architekturen der Softwaretechnik, 1. Aufl., Wiesbaden: Vieweg+Teubner Verlag

**Hall, M./Brown, L./Chaikin, Y. (2008):** Advanced technologies, / Marty Hall; Larry Brown ; 2, 2nd ed., Upper Saddle River, NJ: Prentice Hall

**Kusay-Merkle, U. (2018):** Agiles Projektmanagement im Berufsalltag, Für mittlere und kleine Projekte, Berlin: Springer Gabler

**Leimeister, J. M. (2020):** Service Management und Service Operations, in: Dienstleistungsengineering und -management, (Hrsg.: Leimeister, J. M.), 2nd ed. 2020, Berlin, Heidelberg: Springer Berlin Heidelberg, S. 279–342

**Linz, T. (2017):** Testen in Scrum-Projekten. Leitfaden für Softwarequalität in der agilen Welt, Aus- und Weiterbildung zum ISTQB® Certified Agile Tester - Foundation Extension, 2nd ed., Heidelberg: dpunkt.verlag

**McKenna, D. (2016):** The art of scrum, How scrum masters bind dev teams and unleash agility, New York/Aliquippa, Pennsylvania: Apress; CA Press

**Rance, S. (2011):** ITIL, Service Transition, 2011 ed., 2. impr, London: TSO The Stationery Office

**Wenz, C. (2008):** JavaScript und Ajax, Das umfassende Handbuch ; [Einführung, Praxis, Referenz ; browserübergreifende Lösungen ; Web 2.0: DOM, CSS, XML, Web Services ; neu in der 8. Auflage: Microsoft Silverlight, ASP.NET AJAX 1.0, Ausblick auf Firefox 3 und JavaScript 1.8, 8., aktualisierte und erw. Aufl., Bonn: Galileo Press

**Xu, L. D./Chaudhry, S. S./Tjoa, A. M. (2008):** Research and Practical Issues of Enterprise Information Systems II, Volume 2 IFIP TC 8 WG 8.9 International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2007) October 14-16, 2007, Beijing, China, 255. Jg., Boston, MA: International Federation for Information Processing

**Mühlbacher, D.(2015):** Das Q in Agile, https://www.sogeti.de/globalassets/germany/download/veroffentlichungen-von-fachbeitragen/muehlbauer\_ots\_-agility\_15-4.pdf, Abruf: 15.10.2020

1. Vgl. Marrone/Kolbe 2011, S. 5 [↑](#footnote-ref-1)
2. Vgl. Gross 2008, S. 38 [↑](#footnote-ref-2)
3. Vgl. Leimeister 2020, S. 299 ff. [↑](#footnote-ref-3)
4. Vgl. Rance 2011, S. 89 f. [↑](#footnote-ref-4)
5. Vgl. Rance 2011, S. 93 [↑](#footnote-ref-5)
6. Vgl. Rance 2011, S. 308 f. [↑](#footnote-ref-6)
7. Vgl. Rance 2011, S. 94 [↑](#footnote-ref-7)
8. Vgl. Xu/Chaudhry/Tjoa 2008, S. 1142 f. [↑](#footnote-ref-8)
9. Vgl. Goll 2011, S. 316 [↑](#footnote-ref-9)
10. Vgl. Goll 2011, S. 336 [↑](#footnote-ref-10)
11. Vgl. Goll 2011, S. 311 [↑](#footnote-ref-11)
12. Vgl. Goll 2011, S. 258 f. [↑](#footnote-ref-12)
13. Vgl. Hemmrich/Harrant 2016, S. 24 [↑](#footnote-ref-13)
14. Vgl. Agiles Projektmanagemnt für dummies, Kap.1 , Abs. 1 [↑](#footnote-ref-14)
15. http://projektmanagement-definitionen.de/glossar/agiles-projektmanagement/ [↑](#footnote-ref-15)
16. Vgl. Fowler 2019, S. 5 [↑](#footnote-ref-16)
17. Vgl. Crispin/Gregory 2009, S. 25 ff.; Vgl. Foegen/Kaczmarek 2016, S. 30 f. [↑](#footnote-ref-17)
18. Vgl. Kusay-Merkle 2018, S. 40 [↑](#footnote-ref-18)
19. Vgl. McKenna 2016, S. 30 f. [↑](#footnote-ref-19)
20. Vgl. Fowler 2019, S. 60 [↑](#footnote-ref-20)
21. Vgl. McKenna 2016, S. 28 f. [↑](#footnote-ref-21)
22. Vgl. Fowler 2019, S. 83 [↑](#footnote-ref-22)
23. Vgl. Fowler 2019, S. 90 [↑](#footnote-ref-23)
24. Vgl. Foegen/Kaczmarek 2016, S. 60 f. [↑](#footnote-ref-24)
25. Vgl. Dalton 2019, S. 157 [↑](#footnote-ref-25)
26. Vgl. McKenna 2016, S. 33 [↑](#footnote-ref-26)
27. Vgl. Foegen/Kaczmarek 2016, S. 61 [↑](#footnote-ref-27)
28. Vgl. Fowler 2019, S. 27 f. [↑](#footnote-ref-28)
29. Vgl. Fowler 2019, S. 28 f. [↑](#footnote-ref-29)
30. Vgl. McKenna 2016, S. 28 [↑](#footnote-ref-30)
31. Vgl. Foegen/Kaczmarek 2016, S. 64 [↑](#footnote-ref-31)
32. Vgl. Fowler 2019, S. 43 [↑](#footnote-ref-32)
33. Vgl. Linz 2017, S. 7 [↑](#footnote-ref-33)
34. Vgl. Crispin/Gregory 2009, S. 7 f. [↑](#footnote-ref-34)
35. Vgl McKenna 2016, S. 33; Fowler 2019, S. 48 [↑](#footnote-ref-35)
36. Vgl. Linz 2017, S. 6 [↑](#footnote-ref-36)
37. https://www.gruenderszene.de/lexikon/begriffe/html [↑](#footnote-ref-37)
38. https://developer.mozilla.org/de/docs/Web/HTML [↑](#footnote-ref-38)
39. https://www.heise.de/tipps-tricks/HTML-Grundlagen-Was-Einsteiger-wissen-muessen-3887124.html [↑](#footnote-ref-39)
40. https://www.hanseranking.de/html5.html [↑](#footnote-ref-40)
41. https://developer.mozilla.org/de/docs/Learn/Getting\_started\_with\_the\_web/CSS\_basics [↑](#footnote-ref-41)
42. Architecting CSS, ISBN:978-1-4842-5750-0, S.1-2 [↑](#footnote-ref-42)
43. http://page.mi.fu-berlin.de/mbudde/css\_kurs/vorteile.html [↑](#footnote-ref-43)
44. https://www.hessendscher.de/benefits/index07.htm [↑](#footnote-ref-44)
45. Wenz 2008, S. 31 [↑](#footnote-ref-45)
46. Wenz 2008, S. 21 [↑](#footnote-ref-46)
47. Hall/Brown/Chaikin 2008, S. 23 [↑](#footnote-ref-47)
48. https://www.thymeleaf.org/ [↑](#footnote-ref-48)
49. Vgl. Jarosch H. (2016) S. 20 [↑](#footnote-ref-49)
50. Vgl. Jarosch H. (2016) S.19 [↑](#footnote-ref-50)
51. https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/ [↑](#footnote-ref-51)
52. https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/ [↑](#footnote-ref-52)
53. https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/ [↑](#footnote-ref-53)
54. https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/ [↑](#footnote-ref-54)
55. https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/ [↑](#footnote-ref-55)
56. https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/ [↑](#footnote-ref-56)
57. https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/ [↑](#footnote-ref-57)
58. https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/ [↑](#footnote-ref-58)
59. https://entwickler.de/online/datenbanken/grundkurs-datenbanken-datenbanksysteme-579859345.html [↑](#footnote-ref-59)
60. https://entwickler.de/online/datenbanken/grundkurs-datenbanken-datenbanksysteme-579859345.html [↑](#footnote-ref-60)
61. Wolf 2019, S. 3 [↑](#footnote-ref-61)
62. **Mühlbacher, D. 2015** [↑](#footnote-ref-62)