Abschlussprüfung Sommer 2018

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

"Single-Sign-On"-Authentifizierungsserver mit Beispielimplementierung eines Clients

Abgabetermin: Berlin, den 12.04.2018

Prüfungsbewerber:
Sibylle Blümke
Scharnweberstr. 6
10247 Berlin

wycomco.

Ausbildungsbetrieb:
wycomco GmbH
Fasanenstr. 35
10719 Berlin

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AD	Active Directory
API	Application Programming Interface
CSS	Cascading Style Sheets
GUI	Graphical User Interface
HTML	Hypertext Markup Language
PHP	PHP: Hypertext Preprocessor
SSO	Single Sign On
SQL	Structured Query Language
TDD	Test Driven Development
UML	Unified Modeling Language

1 Einleitung

Die folgende Projektdokumentation wurde im Rahmen des IHK Abschlussprojektes erstellt, welches während der Ausbildung zum Fachinformatiker Fachrichtung Anwendungsentwicklung durchgeführt wurde.

1.1 Projektumfeld

Ausbildungsbetrieb ist die WYCOMCO GMBH, im Folgenden als *wycomco* bezeichnet. Wycomco ist ein Full-Service-Dienstleister im IT-Bereich mit Sitz in Berlin. Zu den Produkten des Unternehmens gehören außerdem individuell anpassbare Softwarelösungen für Anforderungen aller Art. Momentan beschäftigt das Unternehmen 12 Mitarbeiter.

Der Auftraggeber des Projektes ist die Entwicklungsabteilung von wycomco.

1.2 Projektziel

Wycomco implementiert immer mehr Webanwendungen zur internen Nutzung. Bisher müssen sich die Mitarbeiter an jeder Anwendung mit eigenem - teils verschiedenen - Benutzernamen und Passwort registrieren und anschließend anmelden. Rechte und Rollen werden manuell gesetzt und eine Anbindung an die AD ist nicht vorhanden.

Mit einem Single-Sign-On Server soll dieser Prozess automatisiert und vereinfacht werden. Die eigenständige Applikation, im folgenden *wy-connect* genannt, verwaltet die Benutzer an zentraler Stelle und vereinheitlicht die Nutzerdaten der verschiedenen Anwendungen. Die Anmeldung erfolgt ausschließlich am SSO über einen sicheren Kanal.

1.3 Projektbegründung

Im Unternehmen wird die Softwareentwicklung vorangetrieben und auch die Entwicklung im Eigenbedarf steigt stetig an. So existieren momentan drei Anwendungen die im Betrieb laufen oder in der Testphase sind. Alle drei Apps haben eine eigene Benutzerverwaltung mit eigenen Benutzerkennungen. Wird bei der ersten ein Nutzername zum anmelden vergeben, nutzt die zweite die Firmenmailadresse oder eine ID.

An allen muss in regelmäßigen Abständen das Passwort geändert werden. Damit gehört es zum Standard sich Passwörter aufzuschreiben, Passwörter generieren zu lassen oder vergessene Passwörter neu zu vergeben. Der Einfachheit halber neigt man dazu Trivialpasswörter zu nutzen, diese wiederholt einzusetzen und sie sich an unsicheren Stellen zu notieren[1].

[1] vgl. INTERSOFT CONSULTING SERVICES AG [2016]

Durch den wiederholten Anmeldeprozess an verschiedenen Diensten mit verschiedenen Sicherheitsvorkehrungen steigt zudem die Wahrscheinlichkeit, dass ein Passwort ausgespäht wird. Dazu ein Anwendungsdiagramm im Anhang A.4: Use-Case-Diagramm auf Seite iv.

Durch die Implementierung einer Single Sign-on-Lösung muss sich der Nutzer nur ein Master-Passwort merken und dieses einmalig bei der Anmeldung am wy-connect Dienst eingeben. Bei der Nutzung von SSO ist der größte Vorteil darin zu sehen, dass sich der Nutzer nicht nochmal registrieren muss, so dass das lästige Eintippen von Daten, Festlegen eines neuen Passwortes und Bestätigung der Registrierung entfällt. Auch der administrative Aufwand verringert sich bei einer zentralen Nutzerverwaltungfootnotevgl. GmbH [2016].

Dies hätte auf jeden Fall eine bessere Usability und eine nicht zu unterschätzende Zeitersparnis für Anwender im Vergleich zur derzeitigen Lösung zur Folge. Aufgrund der angeführten Gründe hat sich die wycomco GmbH entschieden die Entwicklung von wy-connect in Auftrag zu geben.

1.4 Projektschnittstellen

Die Anwendung benötigt keinerlei Schnittstellen zu anderen Diensten. Der SSO-Server kommuniziert im Rahmen des Abschlussprojektes nur mit einem Testclient, der eigens implementiert wird.

Integration mit anderen externen Systemen wie dem AD ist nicht Teil der Anforderung.

1.5 Projektabgrenzung

Die Anbindung an einen Client wird in diesem Projekt nur für einen Testclient durchgeführt, die Durchführung für alle bestehenden Anwendungen im Unternehmen gehört nicht zum Projekt. Es wird allerdings eine Dokumentation bereitgestellt. Diese befindet sich im Anhang lalala(kommt noch).

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung standen 70 Stunden zur Verfügung. Diese wurden vor Projektbeginn auf die verschiedenen Phasen des Entwicklungsprozesses aufgeteilt.

Das Ergebnis der groben Zeitplanung lässt sich dem folgendem gestapelten Balkendiagramm entnehmen.

Eine detaillierte Übersicht der Phasen befindet sich im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

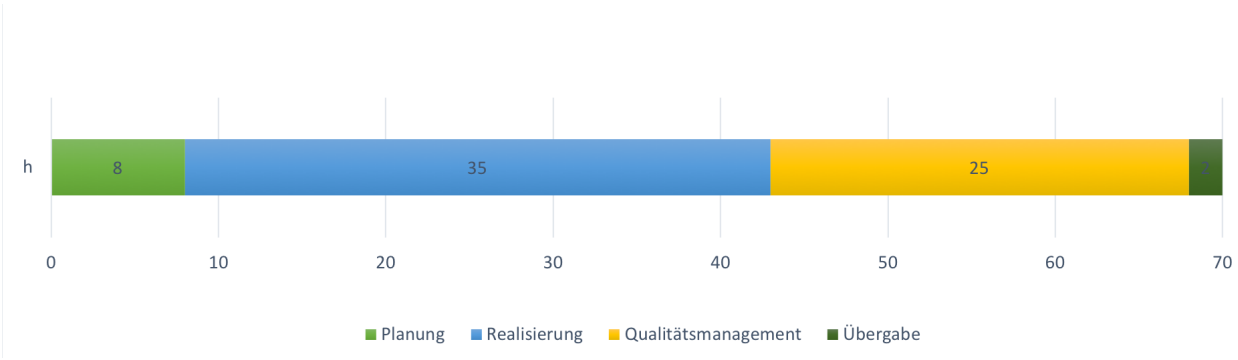


Abbildung 1: grobe Zeitplanung

2.2 Ressourcenplanung

Eine vollständige Auflistung aller während der Umsetzung des Projekts verwendeten Ressourcen befindet sich im Anhang A.2: Verwendete Resourcen auf Seite ii. Bei der Auswahl der verwendeten Software war es wichtig, dass hierdurch keine Zusatzkosten anfallen. Es sollte also Software verwendet werden, die entweder kostenfrei ist (z. B. Open Source), oder für die WYCOMCO bereits Lizenzen besitzt.

2.3 Entwicklungsprozess

Die Vorgehensweise nach dem das Projekt entwickelt wird, nennt sich Entwicklungsprozess. In diesem Fall handelt es sich um das Wasserfallmodell.

Bei diesem nicht iterativen, linearen Prozess werden die einzelnen Projektphasen schrittweise bearbeitet. Hierbei bilden die Phasen-Ergebnisse jeweils die bindende Vorgabe für die nächste Projektphase[2].

Das gesamte Projekt soll mithilfe von Test Driven Development (TDD), zu Deutsch *Testgetriebene Entwicklung*, umgesetzt werden. Das Kernprinzip von TDD besagt, dass das Testen der Programmkomponenten den kompletten Entwicklungsprozess leitet. Es handelt sich hierbei um eine Designstrategie in der das Testen vor der eigentlichen Implementierung stattfindet. Es soll keine Zeile Produktivcode geschrieben werden, die nicht durch einen Test vorher abgedeckt wird. Damit lässt sich die Qualität des Codes erhöhen und den späteren Wartungsaufwand im Nachhinein zu verringern.

Dieses Parallelentwicklung von Code und Tests erfolgt in sich wiederholenden Mikroiterationen, die nur einen kleinen Zeitraum in Anspruch nehmen sollte. Man kann TDD in drei Hauptteile aufspalten, die im englisch Red-Green-Refactor genannt werden. Die einzelnen Phasen lassen sich wie folgt beschreiben:

[2] vgl. [HÜBSCHER U. A. 2007, S. 263]

1. Test schreiben, der fehlschlägt **RED** Im ersten Schritt wird ein Test geschrieben, der die spätere Implementierung auf Funktionalität prüft. Diese Prüfung sollte auch tatsächlich durchgeführt werden um sicherzustellen, dass sich der Test korrekt verhält. Falls ein Test diese Phase bereits ohne Fehler durchläuft, ist er fehlerhaft, da noch keinerlei Implementierung existiert.

2. Minimum an Produktivcode schreiben, dass der Test erfolgreich durchläuft **GREEN** Nach dem fehlschlagen des geschriebenen Tests, wird nur soviel Code geschrieben, dass der Test fehlerfrei durchläuft. Das bedeutet, dass der zu testende Wert oftmals hartkodiert implementiert wird. Damit müssen weitere Test geschrieben werden, dass das hartkodierte Ergebnis nicht mehr ausreicht.

3. Falls nötig – Refaktorisieren des Codes **REFACTOR** Beim Refactoring werden die Tests und der Produktivcode gesäubert. Damit soll die Software einfach, verständlich und frei von Wiederholungen sein. In dieser Phase darf kein neues Verhalten eingeführt werden, was nicht durch einen Test abgedeckt wird. Dabei ist es auch möglich, dass diese Phase erst nach mehrmaligen Durchlaufen der vorherigen beiden Phasen erreicht wird.

Zur Verdeutlichung der drei Phasen die Abbildung 2

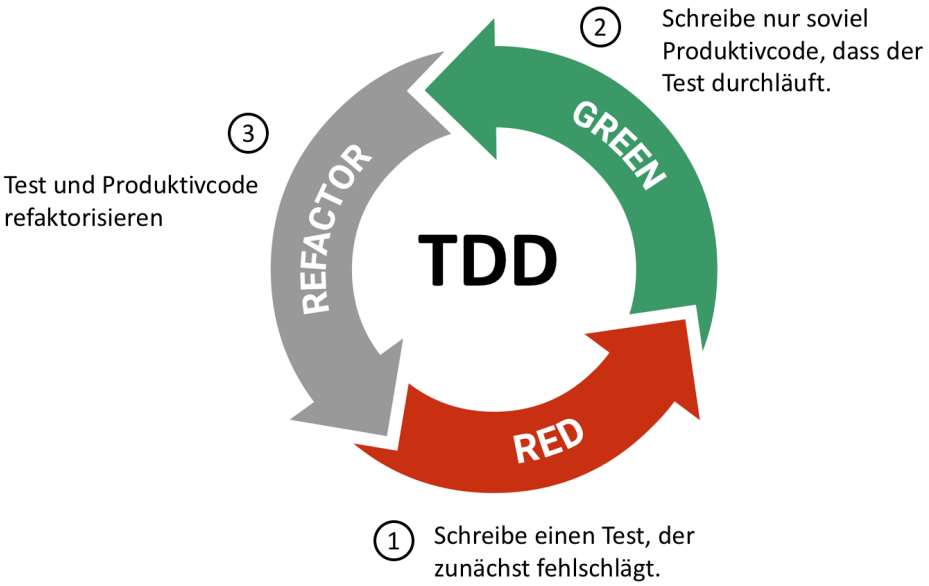


Abbildung 2: TDD Zirkel

3 Analysephase

3.1 Ist-Analyse

Da in dieser frühen Entwicklungsphase noch keine Entwicklertools existieren, muss der Großteil der Testdaten von Hand in der Datenbank angelegt werden. Hierduch kann es passieren, dass bei der Konfiguration der teils sehr komplexen Zusammenhänge und Abhängigkeiten der Datensätze ein Fehler unterläuft. Dies hat zur Folge, das inkonsistente Daten vorliegen, die zu Fehlern im Programmablauf führen können.

Wie in den Abschnitten 1.2 (Projektziel) und 1.3 (Projektbegründung) beschrieben, existiert kein automatisierter Weg eine Datenbank auf inhaltliche Integrität zu prüfen. Die Entwickler müssen die Datenbank manuell validieren. Dieser Prozess sieht in der Regel wie folgt aus:

1. Die betroffene Datenbank in einem GUI-Programm wie *pgAdmin* öffnen
2. Sämtliche betroffenen Felder und Tabellen manuell prüfen
3. Bei komplexeren Regeln muss zusätzlich noch geprüft werden, ob alle benötigten Einträge in den betroffenen Verbindungstabellen existieren

Dieser Prozess ist zeitaufwändig, monoton und gerade bei komplexeren Regeln sehr fehleranfällig. Die Fehlersuche wird hierdurch unnötig verlängert, da zuerst bestimmt werden muss, ob sich der Fehler in der Programmlogik oder in der Datenbank befindet.

3.2 Wirtschaftlichkeitsanalyse

Durch den sehr hohen zeitlichen Mehraufwand, der durch den momentan Prozess anfällt, ist eine Umsetzung des Projektes unbedingt nötig. In den folgenden Abschnitten wird erläutert, ob sich das Projekt auch aus wirtschaftlicher Sicht für das Unternehmen lohnt.

3.2.1 „Make or Buy“-Entscheidung

Da es sich bei der Problematik um ein sehr spezifisches Problem der Entwicklung von *syn6* handelt, konnten die Entwickler kein geeignetes Tool auf dem Markt finden, was allen benötigten Ansprüchen entspricht. Darum wurde sich dazu entschieden, das Projekt in Eigenentwicklung durchzuführen.

3.2.2 Projektkosten

Bei der Berechnung der Projektkosten müssen sowohl die Personalkosten, die durch die Entwicklung des Projektes anfallen, als auch die verwendeten Ressourcen berücksichtigt werden. Bei den Personalkosten muss weiterhin zwischen den Stundensätzen von Auszubildenden und Mitarbeitern unterschieden werden.

Der Stundensatz eines Mitarbeiters wird mit 37,50 € bemessen, der eines Auszubildenden mit 10 €. Für die Nutzung der Ressourcen[3] wird ein pauschaler Stundensatz von 15 € angenommen. Sämtliche Angaben stammen aus dem Controlling.

Aus diesen Werten ergeben sich die Projektkosten in Tabelle 1.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	10 € + 15 € = 25 €	1750,00 €
Fachgespräch	3 h	37,50 € + 15 € = 52,50 €	157,50 €
Code-Review	2 h	37,50 € + 15 € = 52,50 €	105,00 €
Abnahme	1 h	37,50 € + 15 € = 52,50 €	52,50 €
		Projektkosten gesamt	**2065,00 €**

Tabelle 1: Kostenaufstellung

3.2.3 Amortisationsdauer

Die Automatisierung des Validierungsprozesses hat eine deutliche Zeitersparnis zur Folge.

Da es sich nicht um einen täglichen Prozess handelt, und die Dauer einer manuellen Validierung stark abhängig von der Komplexität der zu prüfenden Regel ist, ist es schwer eine Aussage über die tatsächliche Zeitersparnis zu treffen. Schätzungen der Entwickler lagen bei einer Einsparung von 15-60 Minuten pro Prüfung. Für die Berechnung der Amortisationsdauer wird davon ausgegangen, dass eine Prüfung einmal in der Woche nötig ist. Für die Zeitersparnis pro Prüfung werden als Mittelwert 37,5 Minuten angenommen.

Dies ergibt eine tägliche Ersparnis von

$$\frac{37,5 \text{ min/Woche}}{5 \text{ Tage/Woche}} = 7,5 \text{ min/Tag} \tag{1}$$

Bei einer Zeiteinsparung von 7,5 Minuten pro Tag für beide Backend Entwickler und 254 Arbeitstagen[4] im Jahr ergibt sich eine Zeiteinsparung von

$$2 \cdot 254 \frac{Tage}{Jahr} \cdot 7,5 \frac{min}{Tag} = 3810 \frac{min}{Jahr} \approx 63,5 \frac{h}{Jahr} \tag{2}$$

[3] Räumlichkeiten, Arbeitsplatz, etc.
[4] vgl. http://www.schnelle-online.info/Arbeitstage/Anzahl-Arbeitstage-2015.html

Dadurch ergibt sich eine jährliche Einsparung von

$$63,5\ \text{h} \cdot (37{,}5 + 15)\ €/\text{h} = 3333{,}75\ € \tag{3}$$

Die Amortisationszeit beträgt also $\frac{2065{,}00\ €}{3333{,}75\ €/\text{Jahr}} \approx 0,6$ Jahre ≈ 7 Monate.

3.3 Nutzwertanalyse

Neben den in 3.2.3 (Amortisationsdauer) aufgeführten wirtschaftlichen Vorteilen ergeben sich durch Realisierung des Projekts noch einige zusätliche nicht-monitäre Vorteile.

Ein automatisierter Prüfprozess ermöglicht es, das Ergebnis einer Prüfung weiterzuvearbeiten, um es z. B. in einer Weboberfläche oder einem Monitoring System anzeigen zu lassen. Außerdem ermöglicht es die Einbindung in das **CI!**-System um die Datenbank kontinuierlich zu Prüfen.

Obwohl das Projekt im Rahmen von *syn6* entwickelt wurde, lässt es sich auch Problemlos auf andere Projekte und Datenbanken Anwenden. Der tatsächliche Nutzen geht also über *syn6* hinaus.

3.4 Qualitätsanforderungen

Die Qualitätsanforderungen an die Anwendung lassen sich Tabelle 2 entnehmen.

Qualitätsmerkmal	Definition
Abgabetermin einhalten	Der Abgabetermin vom 08.12.2015 ist einzuhalten.
Datenbankagnostisch	Die Anwendung muss unabhängig des verwendeten **RDBMS!** lauffähig sein.
Konsolenbasiert	Die Anwendung muss über das Kommandozeileninterface bedienbar sein.
Wartbarkeit	Um die Wartbarkeit der Anwendung durch die Mitarbeiter von SYNECTIC zu gewährleisten, muss bei der Auswahl der Technologien darauf geachtet werden, dass diese in der Firma vertraut sind (z. B. Auswahl der Programmiersprache).

Tabelle 2: Qualitätsanforderungen

3.5 Lastenheft/Fachkonzept

- Auszüge aus dem Lastenheft/Fachkonzept, wenn es im Rahmen des Projekts erstellt wurde.
- Mögliche Inhalte: Funktionen des Programms (Muss/Soll/Wunsch), User Stories, Benutzerrollen

Beispiel Ein Beispiel für ein Lastenheft findet sich im Anhang A.3: Lastenheft (Auszug) auf Seite iii.

3.6 Zwischenstand

Tabelle 3 zeigt den Zwischenstand nach der Analysephase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Analyse des Ist-Zustands	3 h	2 h	-1 h
2. Wirtschaftlichkeitsanalyse	1 h	2 h	+1
3. Erstellen des Lastenhefts	3 h	3 h	

Tabelle 3: Zwischenstand nach der Analysephase

4 Entwurfsphase

4.1 Zielplattform

Wie in Abschnitt 1.2 (Projektziel) erwähnt, soll am Ende des Abschlussprojektes eine eigenständige Konsolenanwendung vorliegen. Die Anwendung muss vorerst nur auf Linuxsystemen lauffähig sein, da dies auch die Zielplattform von *syn6* ist.

Als Programmiersprache wurde Scala[5] gewählt. Dies bot sich an, da sowohl der Autor, als auch die Backend-Entwickler SYNECTICs mit dieser Sprache bereits vertraut sind. Dies erleichtert später die Wartung des Projektes.

Für das Dateiformat der Regeldateien wurde die **HOCON!** (**HOCON!**)[6] gewählt. Die Nutzwertanalyse, die der Entscheidung für das Dateiformat zugrunde liegt, lässt sich Tabelle 4 entnehmen. Sowohl für die Gewichtung als auch für die Bewertung der einzelnen Kriterien wurde eine Skala von 1 bis 5 verwendet[7][8].

4.2 Architekturdesign

Um die Abhängigkeiten zwischen den einzelnen Programmkomponenten zu verwalten, wurde das Cake Pattern verwendet[9]. Beim Cake Pattern werden mithilfe von Scalas Traits Module erstellt, die dann als Mixin in anderen Komponenten verwendet werden können. Ein Vorteil gegenüber den meisten **DI!** (**DI!**) Frameworks ist, dass hierbei bereits zur Kompilierungszeit festgestellt werden kann, ob alle benötigten Abhängigkeiten erfüllt sind.

[5] http://www.scala-lang.org
[6] https://github.com/typesafehub/config/blob/master/HOCON.md
[7] Gewichtung: 1 = unnötig, 5 = unbedingt erforderlich
[8] Bewertung: 1 = mangelhaft, 5 = sehr gut
[9] vgl. http://lampwww.epfl.ch/~odersky/papers/ScalableComponent.pdf

Eigenschaft	Gewichtung	JSON	YAML	XML	CSV	HOCON
Scala Libraries	5	5	2	5	3	5
Implementierung	4	4	1	3	5	4
Lesbarkeit	4	4	5	2	1	5
Dateigröße	1	3	4	1	5	5
Multline Support	4	2	5	5	1	5
Gesamt	**18**	**68**	**58**	**66**	**48**	**86**
Nutzwert		**3,78**	**3,22**	**3,67**	**2,67**	**4,78**

Tabelle 4: Nutzwertanalyse Dateiformat

Zur Kommunikation mit der zu testenden Datenbank wurde das *Slick* Framework verwendet[10]. Da die Anwendung nur reine SQL Queries ausführen muss, und daher kein **ORM!** (**ORM!**) benötigt wird, wird das Slick Framework hauptsächlich als dünner **JDBC!** (**JDBC!**)-Wrapper verwendet.

Für das Parsen der Kommandozeilenparameter wurde die Library *scopt*[11] verwendet.

4.3 Entwurf der Benutzeroberfläche

Da die Anwendung automatisiert ausgeführt werden soll, wurde auf eine grafische Benutzeroberfläche verzichtet. Stattdessen soll die Anwendung über die Kommandozeile gestartet werden können.

Wie bei Kommandozeilenprogrammen üblich, soll ein Hilfetext existieren, der die Syntax, sowie sämtliche Paramter, Optionen und Flags enthält.

Als einzige benötigter Parameter ist der Pfad zur Hauptdatei anzugeben. Zusätzlich soll es möglich sein, den Pfad anzugeben in dem sich die in der Hauptdatei referenzierten Regeldateien befinden und den Pfad der Ergebnisdatei anzupassen. Zuletzt soll ein Flag existieren, dass das Ergebnis nicht in eine Datei speichert, sondern in die Konsole ausgibt. Dies soll es ermöglichen, den Output einer Prüfung direkt in andere Skripte zu pipen um es z. B. direkt weiterzuverarbeiten.

4.4 Datenmodell

Im folgenden sollen die wichtigsten Komponenten des Datenmodells genannt und kurz erläutert werden. Ein vollständiges Komponentendiagramm befindet sich im Anhang A.6: Komponentendiagramm auf Seite vi.

Runner Der Runner ist dafür zuständig, auf Grundlage einer Konfigurationsdatei, einen vollständigen Testdurchlauf zu starten. Der Runner selbst besitzt relativ wenig eigene Funktionalität, sondern delegiert stattdessen die einzelnen Aufgaben, wie z. B. das Parsen der Regeln und die Ausgabe des Ergebnisses, an die zuständigen Komponenten.

[10] http://slick.typesafe.com/
[11] https://github.com/scopt/scopt

Regel Eine Regel repräsentiert eine zu prüfende Bedinung. Es wird hierbei zwischen einfachen und Kompositregeln unterschieden. Beide Arten von Regeln besitzen einen Namen und eine Beschreibung. Eine simple Regel besteht zusätzlich aus einem SQL-Query. Dieses Query stellt die Bedingung dar, die durch die Regel validiert werden soll. Eine Kompositregel besitzt kein eigenes Query, sondern stellt eine Verknüpfung aus mehreren anderen Regeln dar. Die verknüpften Regeln können wiederum Kompositregeln sein.

Reporter Der Reporter ist für die Ausgabe des Ergebnisses zuständig.

4.5 Geschäftslogik

Der Ablauf einer Validierung lässt sich in vier Phasen unterteilen:

Einlesen der Hauptdatei Die Hauptdatei ist der Ausgangspunkt einer Prüfung. Sie enthält neben sämtlichen auszuführenden Regeln auch alle benötigten Daten für den Verbindungsaufbau mit der zu testenden Datenbank. Ein Beispiel einer solchen Datei findet sich im Anhang A.8: Beispiel Hauptregeldatei auf Seite viii.

Erstellen der Regeln In dieser Phase werden sämtliche Regeln die in der Hauptdatei referenziert sind erstellt. Hierbei wird anhand des Regelnamens die dazugehörige Datei ermittelt und eingelesen. Anschließend wird die Regeldatei geparst und das korrekte Regelobjekt erstellt. Im Anhang A.7: Aktivitätsdiagramm - Einlesen einer Regel auf Seite vii findet sich ein Aktivitätsdiagramm, dass diesen Vorgang beschreibt.

Evaluieren der Regeln Nun folgt die eigentliche Prüfung. Die Runner Komponente evaluiert sämtliche Regeln die im vorherigen Schritt erstellt wurden und erstellt daraus ein Ergebnisset.

Die Evaluation einer einfachen Regeln besteht daraus, dass das in der Regeln definierte SQL-Query abgesetzt wird. Bei einer Kompositregel werden sämtliche im Komposit definierten Regeln rekursiv evaluiert und anschließend je nach Kompositionsart miteinander verknüpft.

Ergebnisausgabe Der letzte Schritt ist die Ausgabe des Gesamtergebnisses. Der Reporter bekommt hierbei die gesamte Ergebnissmenge übergeben und erstellt daraus eine **XML!**-Datei.

4.6 Pflichtenheft/Datenverarbeitungskonzept

- Auszüge aus dem Pflichtenheft/Datenverarbeitungskonzept, wenn es im Rahmen des Projekts erstellt wurde.

Beispiel Ein Beispiel für das auf dem Lastenheft (siehe Kapitel 3.5: Lastenheft/Fachkonzept) aufbauende Pflichtenheft ist im Anhang A.11: Pflichtenheft (Auszug) auf Seite x zu finden.

4.7 Zwischenstand

Tabelle 5 zeigt den Zwischenstand nach der Entwurfsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Entwurf der Regelsyntax	3 h	1,5 h	-1,5 h
2. Entwurf des Kommandozeileninterfaces	2 h	1 h	-1 h
3. Definition des Ausgabeformats	4 h	4 h	
4. Ersellen eines UML-Komponentendiagramms der Anwendung	4 h	2 h	-2 h
5. Erstellen des Pflichtenhefts	4 h	5 h	+1 h

Tabelle 5: Zwischenstand nach der Entwurfsphase

5 Implementierungsphase

5.1 Implementierung der Datenstrukturen

Die zwei wichtigsten Datentypen, die das Herzstück der Anwendung bilden, sind Regeln und Kompositstrukturen, d.h. logische Verkettungen von Regeln.

Beide Datenstrukturen wurden als **ADT!** (**ADT!**) realisiert, genauer gesagt als *Summentyp*.

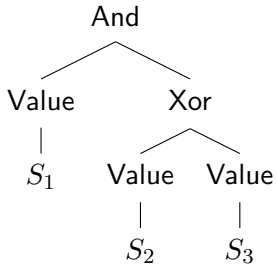
Eine Regel kann zwei mögliche Formen annehmen. Eine *SimpleRule*, d.h. eine einfache Regel, die nur aus Name, Beschreibung und dem auszuführenden SQL-Query besteht, oder eine *CompositeRule*, also eine zusammengesetzte Regel. Zusammgesetzte Regeln besitzen selbst keine SQL-Abfrage, sonder enthalten stattdessen eine Verkettung von Regeln, wobei diese verketteten Regeln wiederum Kompositregeln sein können. Die implementierung des Regel **ADT!** befindet sich im Anhang A.9: Implementierung des Regel **ADT!**s auf Seite viii.

Eine solche Verkettung von Regeln wird durch den *Composition* Datentyp abgebildet. Ein Komposit mehrere Formen annehmen, welches die Art der Verkettung repräsentiert. Die ersten vier Formen sind eine **AND**, **OR**, **XOR** oder **NAND** Verknüpfung. Eine Komposit stellt eine Baumstruktur dar, wobei jeder Zweig des Baums wiederrum ein Komposit enthält. Die Blätter des Baumes enthalten den speziellen Typ **Value** der keine weitere Verkettung, sondern eine *SimpleRule* enthält.

Beispiel Gegeben sei der folgende Ausdruck, wobei S_n eine *SimpleRule* darstellt

$$S_1 \wedge (S_2 \oplus S_3)$$

Daraus würde sich also folgendes Komposit ergeben:



Die Implementierung des *Composition* Datentyps findet sich im Anhang A.10: Implementierung des Komposit **ADT!**s auf Seite ix.

5.2 Implementierung des Kommandozeileninterfaces

- Scopt Library (erstellt automatisch Hilfetext)
- Kommandozeileninterface macht nicht mehr als den Runner mit der CLI Config aufzurufen
- Screenshot des Hilfetextes im Anhang

5.3 Implementierung der Geschäftslogik

- Beschreibung des Vorgehens bei der Umsetzung/Programmierung der entworfenen Anwendung.
- Ggfs. interessante Funktionen/Algorithmen im Detail vorstellen, verwendete Entwurfsmuster zeigen.
- Quelltextbeispiele zeigen.
- Hinweis: Wie in Kapitel 1: Einleitung zitiert, wird nicht ein lauffähiges Programm bewertet, sondern die Projektdurchführung. Dennoch würde ich immer Quelltextausschnitte zeigen, da sonst Zweifel an der tatsächlichen Leistung des Prüflings aufkommen können.

Beispiel Die Klasse `ComparedNaturalModuleInformation` findet sich im Anhang **??**: **??** auf Seite ??.

5.4 Zwischenstand

Tabelle 6 zeigt den Zwischenstand nach der Implementierungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Implementierung des Kommandozeileninterfaces	2 h	3 h	+1 h
2. Implementierung des Regelparsers	13 h	16 h	+3 h
3. Implementierung des Testrunners	12 h	10 h	-2 h
4. Implementierung des Ergebnis Reporters	6 h	5 h	-1 h

Tabelle 6: Zwischenstand nach der Implementierungsphase

6 Abnahmephase

- Welche Tests (z. B. Unit-, Integrations-, Systemtests) wurden durchgeführt und welche Ergebnisse haben sie geliefert (z. B. Logs von Unit Tests, Testprotokolle der Anwender)?
- Wurde die Anwendung offiziell abgenommen?

Beispiel Ein Auszug eines Unit Tests befindet sich im Anhang **??**: **??** auf Seite ??. Dort ist auch der Aufruf des Tests auf der Konsole des Webservers zu sehen.

6.1 Zwischenstand

Tabelle 7 zeigt den Zwischenstand nach der Abnahmephase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Abnahmetest der Fachabteilung	1 h	1 h	

Tabelle 7: Zwischenstand nach der Abnahmephase

7 Einführungsphase

- Welche Schritte waren zum Deployment der Anwendung nötig und wie wurden sie durchgeführt (automatisiert/manuell)?
- Wurden ggfs. Altdaten migriert und wenn ja, wie?
- Wurden Benutzerschulungen durchgeführt und wenn ja, Wie wurden sie vorbereitet?

7.1 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Einführungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Einführung/Benutzerschulung	1 h	1 h	

Tabelle 8: Zwischenstand nach der Einführungsphase

8 Dokumentation

- Wie wurde die Anwendung für die Benutzer/Administratoren/Entwickler dokumentiert (z. B. Benutzerhandbuch, API-Dokumentation)?
- Hinweis: Je nach Zielgruppe gelten bestimmte Anforderungen für die Dokumentation (z. B. keine IT-Fachbegriffe in einer Anwenderdokumentation verwenden, aber auf jeden Fall in einer Dokumentation für den IT-Bereich).

Beispiel Ein Ausschnitt aus der erstellten Benutzerdokumentation befindet sich im Anhang **??**: **??** auf Seite ??. Die Entwicklerdokumentation wurde mittels PHPDoc[12] automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang **??**: **??** auf Seite ??.

8.1 Zwischenstand

Tabelle 9 zeigt den Zwischenstand nach der Dokumentation.

Vorgang	Geplant	Tatsächlich	Differenz
1. Erstellen der Benutzerdokumentation	2 h	2 h	
2. Erstellen der Projektdokumentation	6 h	8 h	+2 h
3. Programmdokumentation	1 h	1 h	

Tabelle 9: Zwischenstand nach der Dokumentation

9 Fazit

9.1 Soll-/Ist-Vergleich

- Wurde das Projektziel erreicht und wenn nein, warum nicht?
- Ist der Auftraggeber mit dem Projektergebnis zufrieden und wenn nein, warum nicht?
- Wurde die Projektplanung (Zeit, Kosten, Personal, Sachmittel) eingehalten oder haben sich Abweichungen ergeben und wenn ja, warum?

[12] Vgl. **?**

- Hinweis: Die Projektplanung muss nicht strikt eingehalten werden. Vielmehr sind Abweichungen sogar als normal anzusehen. Sie müssen nur vernünftig begründet werden (z. B. durch Änderungen an den Anforderungen, unter-/überschätzter Aufwand).

Beispiel (verkürzt) Wie in Tabelle 10 zu erkennen ist, konnte die Zeitplanung bis auf wenige Ausnahmen eingehalten werden.

Phase	Geplant	Tatsächlich	Differenz
Entwurfsphase	19 h	19 h	
Analysephase	9 h	10 h	+1 h
Implementierungsphase	29 h	28 h	-1 h
Abnahmetest der Fachabteilung	1 h	1 h	
Einführungsphase	1 h	1 h	
Erstellen der Dokumentation	9 h	11 h	+2 h
Pufferzeit	2 h	0 h	-2 h
Gesamt	70 h	70 h	

Tabelle 10: Soll-/Ist-Vergleich

9.2 Lessons Learned

- Was hat der Prüfling bei der Durchführung des Projekts gelernt (z. B. Zeitplanung, Vorteile der eingesetzten Frameworks, Änderungen der Anforderungen)?

9.3 Ausblick

- Wie wird sich das Projekt in Zukunft weiterentwickeln (z. B. geplante Erweiterungen)?

Literaturverzeichnis

PHP

PHP Manual. : *PHP Manual*, http://php.net/manual/de/index.php

intersoft consulting services AG 2016

AG intersoft consulting s.: Single Sign-on: Tipps beim Einsatz der Login-Technologie. (2016). https://www.datenschutzbeauftragter-info.de/single-sign-on-tipps-beim-einsatz-der-login-technologie/

Git

Git: *Git Documentation*, https://git-scm.com/documentation

GmbH 2016

GmbH, Univention: Kurz erklärt: Einmalige Anmeldung per Single Sign-on. (2016). https://www.univention.de/2016/12/einmalige-anmeldung-per-single-sign-on/

Hardt

Hardt, D.: *The OAuth 2.0 Authorization Framework*, https://tools.ietf.org/html/rfc6749

Hübscher u. a. 2007

Hübscher, Heinrich ; Petersen, Hans-Joachim ; Rathgeber, Carsten ; Richter, Klaus ; Dr. Scharf, Dirk: *IT-Handbuch*. 5. Westermann, 2007

Oracle

Oracle: *MySQL Documentation*, https://dev.mysql.com/doc/

Otwell a

Otwell, Taylor: *Laravel Documentation*, http://www.laravel.com

Otwell b

Otwell, Taylor: *Laravel Passport*, https://laravel.com/docs/master/passport

Otwell c

Otwell, Taylor: *Laravel Socialite*, https://laravel.com/docs/5.5/socialite

Pages

Pages, GitLab: *GitLab*, https://docs.gitlab.com

Shore 2005

Shore, James: *Red-Green-Refactor*. http://www.jamesshore.com/Blog/Red-Green-Refactor.html, November 2005

Eidesstattliche Erklärung

Ich, Sibylle Blümke, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

> *"Single-Sign-On"-Authentifizierungsserver mit Beispielimplementierung eines Clients –*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Berlin, den 12.04.2018

Sibylle Blümke

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase			**7 h**
1. Analyse des Ist-Zustands		3 h	
1.1. Fachgespräch mit den Projektleitern	1 h		
1.2. Prozessanalyse	2 h		
2. Wirtschaftlichkeitsanalyse durchführen		1 h	
3. Erstellen des Lastenhefts mit den Projektleitern		3 h	
Entwurfsphase			**17 h**
1. Entwurf der Regelsyntax		3 h	
1.1. Auswahl des Dateiformats	1 h		
1.2. Definition der Syntax	2 h		
2. Entwurf des Kommandozeileninterfaces		2 h	
2.1. Aufruf in der Konsole	1 h		
2.2. Benötigte Optionen und Flags	1 h		
3. Definition des Ausgabeformats		4 h	
3.1. Auswahl des Dateiformats	1 h		
3.2. Ausgabesyntax des Ergebnisses	2 h		
3.2. Erstellen einer Beispielausgabe	1 h		
4. Erstellen eines UML-Komponentendiagramms der Anwendung		4 h	
5. Erstellen des Pflichtenhefts		4 h	
Implementierungsphase			**33 h**
1. Implementierung des Kommandozeileninterfaces		2 h	
2. Implementierung des Regelparsers		13 h	
2.1. Parsen von einfachen Regeln	3 h		
2.2. Parsen von verschachtelten Regeln	10 h		
3. Implementierung des Testrunners		12 h	
4. Implementierung des Ergebniss Reporters		6 h	
4.1. Programmierung der allgemeinen Reporter Schnittstelle	1 h		
4.2. Programmierung des standard Reporters	4 h		
Übergabe			**1 h**
1. Abnahme durch die Projektleiter		0,5 h	
2. Einweisung in die Anwendung		0,5 h	
Erstellen der Dokumentation			**12 h**
1. Erstellen der Benutzerdokumentation		2 h	
2. Erstellen der Projektdokumentation		9,5 h	
3. Programmdokumentation		0,5 h	
3.1. Generierung durch ScalaDoc	0,5 h		
Gesamt			**70 h**

A.2 Verwendete Resourcen

A.2.1 Hardware

- Büroarbeitsplatz mit iMac

A.2.2 Software

- OS X 10.13 High Sierra – Betriebssystem
- LucidChart – Anwendung zum Erstellen von UML-Diagrammen
- Creately – Anwendung zum Erstellen von Online-UI-Mockups
- draw.io – Diagramm-Editor (Erweiterung für Google Drive)
- PHP – Programmiersprache
- PHPUnit – Testframework für PHP
- Laravel 5.5 - PHP Framework
- Vue.js - Javascript Framework
- HTML, CSS Webtechnologien
- JetBrains PHP Storm – Entwicklungsumgebung PHP
- git – Verteilte Versionsverwaltung
- Gitlab – Selfhosted Repository Verwaltung
- texmaker – LaTeX Editor
- Sequel Pro – Verwaltungswerkzeug für Datenbanken

A.3 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Single-Sign-On-Server
   1.1. Die Anwendung verfügt über eine eigene Datenbank für die Benutzerdaten.
   1.2. Anzeigen einer Übersichtsseite für autorisierte Clienten und Token mit allen relevanten Informationen zu diesen.
   1.3. Die Authentifizierung muss über einen sicheren Kanal erfolgen.
2. Login Prozess der Client Anwendung
   2.1. Die Client Anwendung verfügt über eine eigene Benutzerdatenbank.
   2.2. Die Authentifizierungsmöglichkeit über wy-connect muss einfach auf andere Clients übertragbar sein.
   2.3. Der Nutzer muss die Client Anwendung zum Zugriff autorisieren.
   2.4. Der Nutzer meldet sich nur am Server mit seinen Nutzerdaten an.
3. Sonstige Anforderungen
   3.1. Der Server sowie der Login Prozess am Client soll über eine GUI intuitiv bedient werden können.
   3.2. Der Server sowie der Client müssen ohne Installation von zusätzlicher Software über einen Webbrowser im Intranet erreichbar sein.
   3.3. Zur Versionskontrolle der Anwendungsentwicklung soll ein Git[13]-Repository verwendet werden.
   3.4. Die Anwendung soll in der Programmiersprache PHP[14] mittels des Frameworks Laravel[15] umgesetzt werden.
   3.5. Der Server soll auf dem firmeninternen Webserver gehostet werden.
   3.6. Bei Einsatz einer MySQL-Datenbank[16] soll der firmeninterne MySQL Server Verwendung finden.
   3.7. Der Einrichtungsprozess für neue Clienten muss dokumentiert werden.

[...]

[13] vgl. Git
[14] vgl. PHP
[15] vgl. Otwell [a]
[16] vgl. Oracle

A.4 Use-Case-Diagramm

Das folgende Diagramm beschreibt den Anmeldeprozess an verschiedenen Systemen ohne SSO.

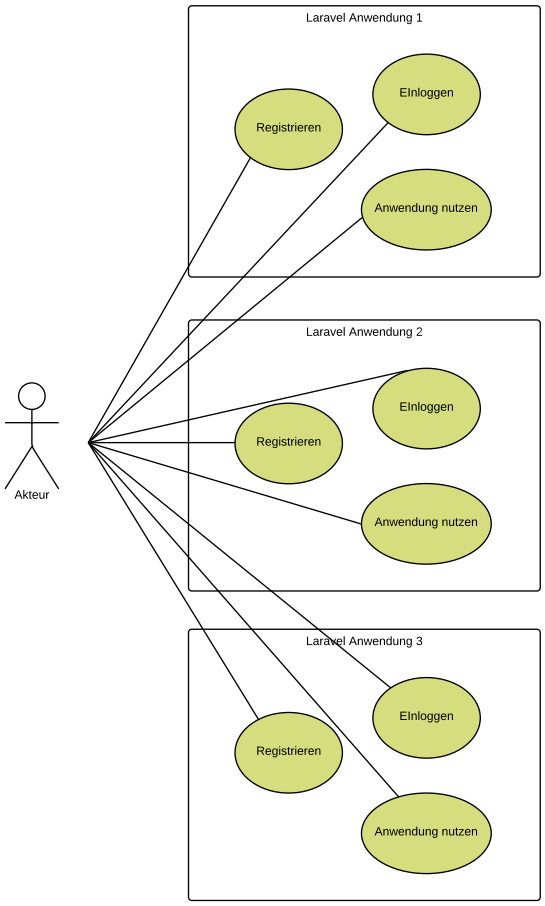


Abbildung 3: Use-Case-Diagramm

A.5 Kommandozeileninterface

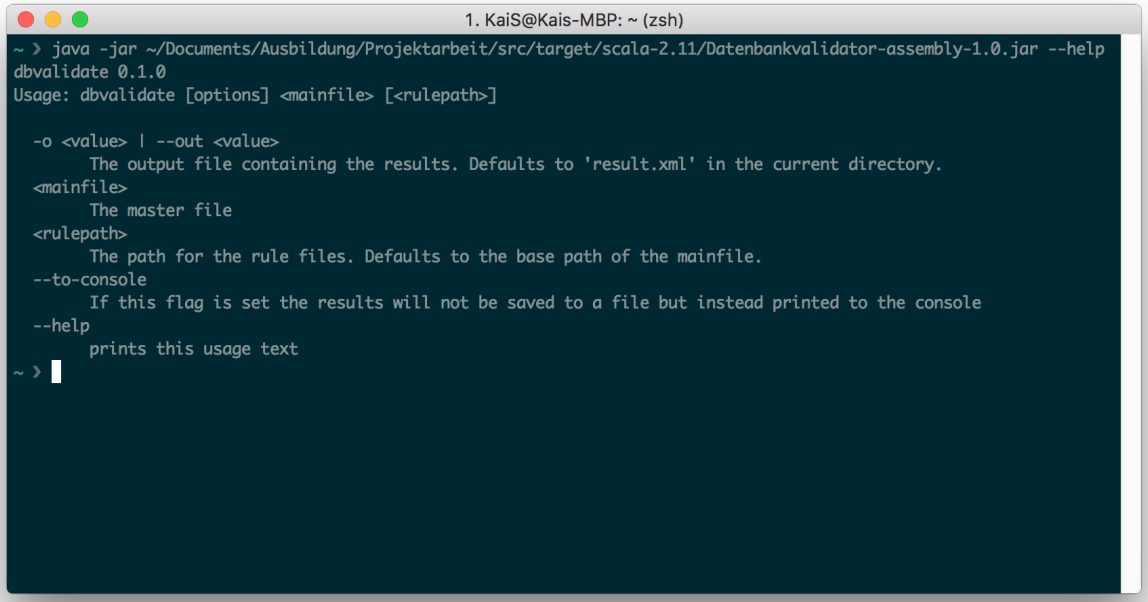


Abbildung 4: Kommandozeileninterface

A.6 Komponentendiagramm

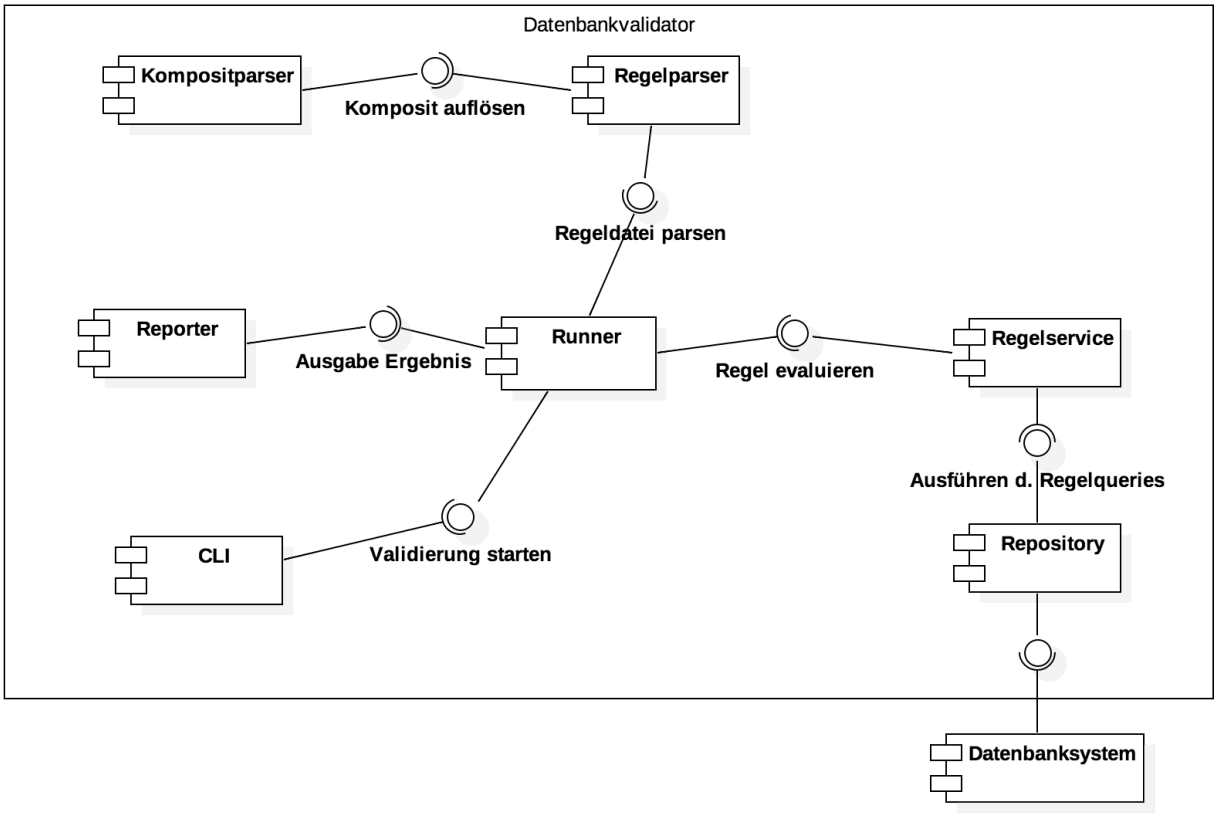


Abbildung 5: Komponentendiagramm

A.7 Aktivitätsdiagramm - Einlesen einer Regel

Das folgende Diagramm beschreibt das Einlesen einer Regel aus einer Regeldatei.

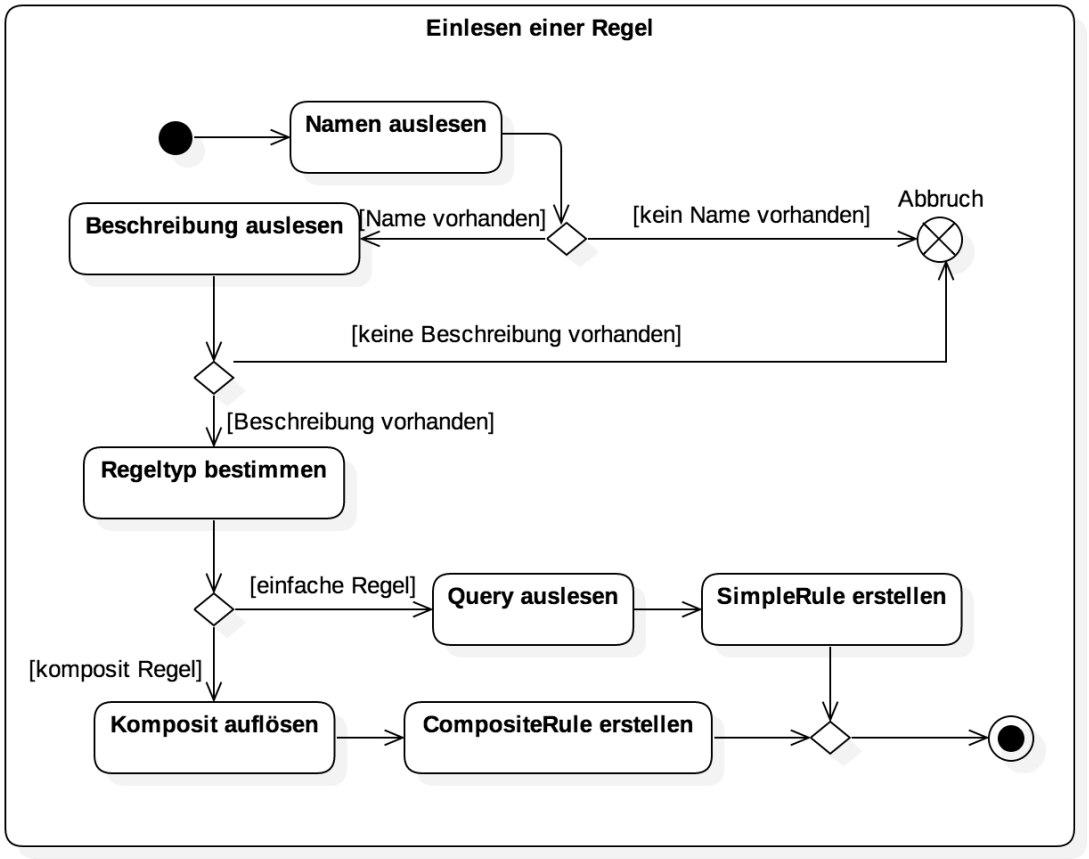


Abbildung 6: Einlesen einer Regel

A.8 Beispiel Hauptregeldatei

Ein Beispiel einer Hauptdatei. In der Hauptdatei werden sowohl die Datenbankverbindungsparamer beschrieben als auch sämtliche Regeln aufgelistet, die evaluiert werden sollen.

```
database {
    driver = "org.sqlite.JDBC"
    url = "jdbc:sqlite:/Users/KaiS/Projects/Abschlussprojekt/src/db.db"
    user = ""
    password = ""
}

rules = [
    one_greater_than_two,
    two_greater_than_one,
    first_and_second,
    first_or_second
]
```

A.9 Implementierung des Regel ADT!s

```
package Rule

/**
 * An ADT representing a rule.
 *
 * A rule can either be a SimpleRule containing only the query through
 * which it is defined, or a CompositeRule which itself does not contain
 * a query, but a composition of rules.
 */
sealed trait Rule {
  def name: String
  def description: String
}

case class SimpleRule(name: String, description: String, query: String) extends Rule
case class CompositeRule(name: String, description: String, composition: Composition) extends Rule
```

A.10 Implementierung des Komposit ADT!s

```
1  package Rule
2
3  /**
4   * An ADT representing a composition of rules.
5   *
6   * Each branch of the composition is itself a composition. A composition
7   * containing only a single rule represents a leaf of the tree and is called
8   * a Value.
9   */
10 sealed trait Composition
11
12 case class And(left: Composition, right: Composition) extends Composition
13 case class Or(left: Composition, right: Composition) extends Composition
14 case class NAnd(left: Composition, right: Composition) extends Composition
15 case class XOr(left: Composition, right: Composition) extends Composition
16 case class Value(rule: Rule) extends Composition
```

A.11 Pflichtenheft (Auszug)

Die geplante Umsetzung der im Lastenheft (Auszug siehe A.3) definierten Anforderungen wird in folgendem Auszug aus dem Pflichtenheft beschrieben:

Umsetzung der Anforderungen

1. Single-Sign-On-Server
   1.1. Für das speichern der Benutzerdaten wird die firmeninterne MySQL Datenbank genutzt.
   1.2. Innerhalb des **SSO-Servers!** gibt eine View für die Benutzerverwaltung:
      - Admins können Nutzer anlegen, löschen, bearbeiten
      - Benutzer können nur ihr eigenes Profil löschen und bearbeiten
   1.3. Als Authentifizierungsprotokoll wird OAuth2 [17]genutzt.
      - die Autorisierung wird hierbei über die API vorgenommen
      - die Authentifizierung erfolgt ausschließlich über den Server
      - zur Umsetzung wird das Laravel Paket Passport[18] genutzt
2. Login Prozess der Client Anwendung
   2.1. Für das speichern der Benutzerdaten wird die firmeninterne MySQL Datenbank genutzt.
   2.2. Die wy-connect Anbindung an andere Laravel Anwendungen wird mittels Installation eines bereitgestellten Pakets umgesetzt.
      - dieses Paket kann über einen einfachen Konsolenbefehl in den Clienten integriert werden
      - der wy-connect Provider liegt dabei auf der firmeneigenen Gitlab Instanz
      - zur Umsetzung wird das Laravel Paket Socialite[19] genutzt
   2.3. Nachdem Login Prozess am SSO wird der Benutzer gefragt ob der Test Client auf die Nutzerdaten zugreifen darf.
   2.4. Innerhalb des Login Prozesses am Clienten werden die Nutzerdatzen nur vom SSO-Server abgefragt.
3. Sonstige Anforderungen
   3.1. Über das Webinterface kann sich ein Nutzer anmelden und hat über eine übersichtlich gestaltete GUI Zugriff auf seine Einstellungen.
   3.2. Das Programm läuft als Webanwendung.
   3.3. Zur Versionskontrolle wird der firmeneigene GitLab-Server[20] genutzt.
   [...]

[17] vgl. Hardt
[18] vgl. Otwell [b]
[19] vgl. Otwell [c]
[20] vgl. Pages