

Hochschule Merseburg
University of Applied Sciences
Fachbereich Informatik und Kommunikationssysteme
Technische Redaktion und E-Learning Systeme



Bachelorarbeit

Prototypische Entwicklung einer Webanwendung in Java unter Verwendung des Play Frameworks

Christopher Kutzmann
Matrikel Nr.: 18730

Eingereicht im September 2016

Erstprüfer Prof. Dr. rer. pol. Uwe Schröter
Zweitprüfer Nico Scheithauer, M.Sc.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Merseburg, _____

Date

Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Merseburg, _____

Datum

Unterschrift

Kurzfassung

Die vorliegende Bachelorarbeit untersucht die Webentwicklung in Java unter Verwendung des Play Frameworks, wobei der Fokus auf der prototypischen Entwicklung einer Webanwendung liegt. Diese Webanwendung stellt ein E-Learning System dar, dass es ermöglicht die funktionale Programmiersprache Scala interaktiv und online zu erlernen bzw. zu lehren. Während der Entwicklung mit Play werden aufkommende Vor- und Nachteile analysiert und abschließend kritisch diskutiert. Zudem werden die erforderlichen Grundlagen, der Webentwicklung mit dem Play Framework, zusammengefasst und erklärt.

Schlagwörter

Java, Scala, Play Framework, Webentwicklung, E-Learning, Webanwendung, MVC, Softwaredesign, UIKit, HTML, Objektorientierte Programmierung, Git

Inhaltsverzeichnis

Kurzfassung	III
Abbildungsverzeichnis	VII
Listings	IX
Abkürzungsverzeichnis	X
1 Einleitung	1
1.1 Motivation	1
1.2 Ziel der Arbeit	3
1.3 Schema der Arbeit	3
2 Grundlegende Software und Konzepte	5
2.1 Die Programmiersprache Java im Web	6
2.1.1 Grundlegende Eigenschaften und Konzepte	6
2.1.2 Java in der Webentwicklung	9
2.2 Das Webframework Play	14
2.2.1 Grundlagen und Struktur	14
2.3 Das Front-End Framework UIKit	21
2.3.1 Grundlagen und Entstehung	21
2.3.2 Bestandteile des Frameworks	21
2.3.3 Anwendungsbeispiel	23
2.4 Softwareversionierung mit Git	27
2.4.1 Entstehung	27

Inhaltsverzeichnis

2.4.2	Grundlagen	27
2.4.3	Allgemeiner Arbeitsablauf	31
3	Theoretischer Entwurf der Webanwendung	33
3.1	Zentrale Anforderungen an die Webanwendung	34
3.1.1	Funktionale Anforderungen	34
3.1.2	Nichtfunktionale Anforderungen	37
3.1.3	Tabellarische Zusammenfassung	38
3.2	Identifikation der funktionalen Komponenten	40
3.3	Aufbau und Funktionsweise der Webanwendung	44
3.3.1	Editor Komponente	46
3.3.2	Kursübersicht	50
3.3.3	Authentifizierung	51
3.3.4	Administration	53
4	Prototypische Implementierung der Webanwendung	55
4.1	Werkzeuge und Entwicklungsumgebungen	56
4.1.1	Server	56
4.1.2	Entwicklungsumgebung	57
4.2	Architektur mit Play	59
4.2.1	Models	59
4.2.2	Controllers	63
4.2.3	Views	66
4.3	Grundlegende Klassen	71
4.3.1	Klasse: Runsc	71
4.3.2	Klasse: ProcessMessage	74
5	Zusammenfassung	76
5.1	Auswertung der Entwicklung	76
5.2	Ausblick	78
	Anhang	80
A	Installation und Inbetriebnahme der Webanwendung	80

Inhaltsverzeichnis

A.1	Vorraussetzungen für die Projektaufnahme und den Betrieb . . .	80
A.2	Anleitung zur Projektaufnahme und Inbetriebnahme	81
B	Beschreibung der Verzeichnisstruktur	82
B.1	Das app/-Verzeichnis	83
B.2	Das models/-Verzeichnis	84
B.3	Das views/-Verzeichnis	85
B.4	Das controllers/-Verzeichnis	86
B.5	Das classes/-Verzeichnis	87
B.6	Das conf/-Verzeichnis	88
B.7	Das project/-Verzeichnis	89
B.8	Das public/-Verzeichnis	90
B.9	Das test/-Verzeichnis	91
B.10	Sonstige Ordner und Dateien	92
C	Initailtemplate Play Framework	93
C.1	Beispielhaftes Java Initialtemplate des Play Frameworks	93
	Literaturverzeichnis	95

Abbildungsverzeichnis

2.1	Java-Kompilierungsprozess	6
2.2	Klassenvererbung	8
2.3	Webanwendung-Prozesse	10
2.4	Modell-View-Controller	13
2.5	Hot-Code-Reload mit beispielhafter Fehlermeldung	15
2.6	Verzeichnisbaum einer MVC-Struktur	17
2.7	HTML-Form ohne UIKit	24
2.8	HTML-Form mit UIKit	25
2.9	Versionierung anhand von Dateienänderung	28
2.10	Versionierung anhand von Schnappschüssen	28
2.11	Zentrale Versionsverwaltung	29
2.12	Verteilte Versionsverwaltung	30
2.13	Braching Workflow	31
3.1	Klassendiagramm - UML	41
3.2	Komponenten Diagramm	44
3.3	Editor (Entwurf)	46
3.4	Popup (Entwurf)	49
3.5	Registration (Entwurf)	52
3.6	Anmeldung (Entwurf)	53
3.7	Administration (Entwurf)	54
4.1	Mapping-Tabelle der Assoziationen (Übungen zu Aufgaben)	61
4.2	Ausgabe eines durch UIKit klassifizierten Fehlertextes	70
4.3	Editor-Benutzeroberfläche mit Quelltext und korrekter Ausgabe	73

Abbildungsverzeichnis

4.4	Die Editor-Benutzeroberfläche mit Quelltext und Fehlerausgabe .	73
B.1	Verzeichnisbaum: scalablecurriculum/	82
B.2	Verzeichnisbaum: app/	83
B.3	Verzeichnisbaum: models/	84
B.4	Verzeichnisbaum: views/	85
B.5	Verzeichnisbaum: controllers/	86
B.6	Verzeichnisbaum: classes/	87
B.7	Verzeichnisbaum: conf/	88
B.8	Verzeichnisbaum: project/	89
B.9	Verzeichnisbaum: public/	90
B.10	Verzeichnisbaum: test/	91
B.11	Verzeichnisbaum: scalablecurriculum/ (Sonstige)	92

Listings

2.1	Annotation in Java	8
2.2	Auszug eines beispielhaften User-Models	18
2.3	Einfaches Scala Template	19
2.4	Objekte innerhalb von Scala Templates	19
2.5	Beispielhafter Controller	20
2.6	Notwendige Routing-Konfiguration	20
2.7	Einbindung von UIKit	23
2.8	HTML-Form ohne UIKit	24
2.9	HTML-Form mit UIKit	25
4.1	Auszug des Model-Objekts Exercise (Übung)	60
4.2	Auszug des Model-Objekts Task (Aufgabe)	60
4.3	Quelltextbeispiel für das Auslesen von Objekten	62
4.4	Quelltextbeispiel für den Aufruf von findByEmail()	62
4.5	SBT Pluginregistrierung des Play Ebean Plugins	63
4.6	Play Ebean Plugin aktivieren	63
4.7	Quelltextbeispiel für den Aufruf von findByEmail()	64
4.8	Quelltextbeispiel für die Routing-Konfiguration	65
4.9	Auszug: app/views/main.scala.html	67
4.10	Auszug: app/views/navigation.scala.html	68
4.11	Auszug: app/views/login.scala.html	69
4.12	Routes-Konfiguration zu Kompilieren & Ausführen	71
4.13	Auszug des EditorControllers (compileCode-Action)	71
4.14	Methode: runProcess()	72
4.15	Wesentlicher Quelltext der ProcessMessage Klasse	74

Abkürzungsverzeichnis

CGI	Common Gateway Interface
CRUD	Creat, Read, Update, Delete
CSS	Cascading Style Sheets
CVS	Concurrent Versions System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrierte Entwicklungsumgebung (eng. Integrated Development Environment)
JPA	Java Persistence API
JRE	Java-Laufzeitumgebung (eng. Java Runtime Environment)
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JVM	Virtuelle Maschine (eng. Java Virtual Machine)
KISS	Keep It Simple, Stupid
LESS	LESS Stylesheet-Sprache
MVC	Model View Controller
OOP	Objektorientierte Programmierung

Listings

ORM	Object-Relational Mapping
RDB	relationale Datenbank
SQL	Structured Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAR	Web Application Archive
XML	Extensible Markup Language

1 Einleitung

1.1 Motivation

Bei der Entwicklung von Webanwendungen in Java greifen Programmierer häufig auf Webframeworks, wie zum Beispiel Spring, Struts2 oder Vaadin zurück. Diese Webframeworks dienen einer komfortableren Entwicklung von dynamischen Webseiten, Webanwendungen oder Webservices. Sie setzen Softwaredesign-Paradigmen wie zum Beispiel »Konvention vor Konfiguration« oder »Keep It Simple, Stupid (KISS)«¹ um, wodurch verschiedenste Sammlungen an strukturierter und gut wartbarer Software den Webentwicklungs-Teams als solide Ausgangsbasis zur Verfügung stehen. Die eben erwähnten Webframeworks bieten ein sehr breites Spektrum an Funktionalität, jedoch ist der Arbeitsaufwand, der auf den Entwickler zukommt, durch zu viel Quelltext und Konfigurationsarbeit sehr groß. Die Vorteile einer Entwicklung mit einem sehr komplexen Webframework zeigen sich in der Regel erst mit steigender Komplexität bzw. steigendem Umfang der zu entwickelnden Webanwendung. Neben den langjährigen etablierten Webframeworks gibt es eine Menge an neuen und innovativen Webframeworks, welche die Webentwicklung in Java mit ihren eigenen Entwicklungskonzepten noch komfortabler und einfacher machen wollen. Das Play Framework ist eines davon. Play verspricht sich von klassischen Webframeworks für die Webentwicklung in Java, im Hinblick auf Ablaufgeschwindigkeit und Entwicklungseffizienz stark zu unterscheiden. Hierfür versucht es einen Kompromiss zwischen zwei verschiedenen Welten zu

¹ Das KISS-Prinzip beschreibt eine Softwareentwicklung unter dem Aspekt, stets die einfachste und eleganteste Lösung umzusetzen (siehe archlinux.de [2, Artikel]).

1 Einleitung

schaffen. Auf der einen Seite wird die Effizienz und Einfachheit von Web-Frameworks, wie zum Beispiel Rails und Skriptsprachen, wie zum Beispiel Ruby angestrebt und auf der anderen Seite eine stabile und sichere Sprache — welche Java darstellt —, als Basis genutzt. Play ist in Java und Scala geschrieben. Ebenso können Play Applikationen in Java und/oder Scala geschrieben werden, was Exkursionen innerhalb der Webentwicklung in Programmierparadigmen, wie zum Beispiel der Funktionalen Programmierung ermöglicht.

Scala ist eine Programmiersprache, die als Multiparadigmen-Sprache bezeichnet wird, da sie sowohl eine objektorientierte als auch eine funktionale Programmiersprache ist. »Scala wird seit 2001 an der École Polytechnique Fédérale de Lausanne (EPFL) von einem Team entwickelt, das Martin Odersky leitet.«² Scala ist eine statisch typisierte Virtuelle Maschine (eng. Java Virtual Machine) (JVM)- und .NET-Sprache mit einer kompakten und modernen Syntax, einem ausgeklügeltem Typsystem und Ausdrucksformen, welche eine Skalierbarkeit von kleinen interpretierbaren Skripten bis hin zu sehr komplexen Anwendungen ermöglichen. Sie wird, wie auch die funktionale Programmierung selbst *mittlerweile* nicht nur im akademischen Raum verwendet. Softwareunternehmen, wie zum Beispiel Novell³ oder Siemens⁴ nutzen Scala zur Realisierung ihrer Softwareprojekte. Ebenso haben renommierte soziale Netzwerke, wie zum Beispiel Twitter ihre Anwendungen mit Scala realisiert⁵. Um Scala gibt es ein großes Angebot an Unterstützung, über die Community (Foren, Mailing Listen oder Chaträume), Literatur, bis hin zu Online Lern-Ressourcen. Wer jedoch Scala gern interaktiv und online, über eine E-Learning Plattform, wie zum Beispiel Codecademy.com oder Code.org lernen möchte, sucht leider vergebens nach einem Angebot.

² siehe Piepmeyer [35, S.30]

³ siehe scala-lang.org [42, Artikel]

⁴ siehe scala-lang.org [41, Forenbeitrag]

⁵ siehe twitter.github.io [51, Artikel]

1.2 Ziel der Arbeit

Ziel der Arbeit ist die prototypische Entwicklung einer Webanwendung in Java unter Verwendung des Play Frameworks. Diese Webanwendung stellt ein minimalistisches E-Learning System dar, mit dem es möglich ist die Programmiersprache Scala interaktiv zu erlernen beziehungsweise zu lehren und ist somit als Lösung, für das im letzten Abschnitt beschriebene noch fehlende Angebot zu verstehen. Es werden die technischen und die didaktischen Möglichkeiten ein solches Curriculum in der Software Entwicklung abzubilden untersucht. Einen wesentlichen Schwerpunkt stellt hierbei die Implementierung, unter Verwendung des Play Frameworks dar. Zu untersuchen sind die Vorteile, welche bei der Entwicklung mit dem Play Framework gegeben sind. Die Webanwendung trägt, in Anlehnung an Scala und einem E-Learning System zugrundeliegendem Curriculum⁶ den Namen »Scalable Curriculum«.

1.3 Schema der Arbeit

Die vorliegende Arbeit weist folgendes Schema auf: Zu Beginn gibt das erste Kapitel eine Einführung in die vorliegende Arbeit. In Kapitel 2 werden folgend grundlegende Konzepte und die zu verwendende Software näher beschrieben, um für die nachfolgenden Kapitel ein Grundverständnis zu schaffen. Dies umfasst eine Einführung in die Verwendung von Java im Web, in die Webentwicklung mit dem Play Framework und in das Front-End Framework UIKit, sowie eine Beschreibung und beispielhafte Verwendung der Versionierungsoftware Git. Das Kapitel 3 befasst sich mit dem theoretischen Entwurf der Webanwendung. Hierbei werden alle zentralen Anforderungen an die Anwendung definiert, alle notwendigen funktionalen Komponenten identifiziert und der allgemeine Aufbau der Webanwendung präsentiert. Aufbauend auf den

⁶ Als ein Curriculum (lateinisch: *Wettlauf, Umlauf*) wird in der Pädagogik ein Lehrplan bzw. Lehrprogramm bezeichnet.

1 Einleitung

Ergebnissen des Entwurfs, beschreibt das Kapitel 4 auszugsweise die Implementierung der Webanwendung. In Kapitel 5, dem letzten Kapitel wird rückblickend über die Implementierung diskutiert und ein allgemeiner Ausblick gegeben.

2 Grundlegende Software und Konzepte

Dieses Kapitel beschreibt die für die Entwicklung der Webanwendung »Scalable Curriculum« verwendete Software und gibt darüber hinaus einen Einblick in einige grundlegende Konzepte der Versionsverwaltung. Grundkenntnisse der hier vorgestellten Software und deren Konzepte sind für das Verständnis der folgenden Kapitel notwendig.

Zu Beginn gibt das Kapitel einen Überblick über die Programmiersprache Java. Neben einer Darstellung der grundlegenden Eigenschaften und Konzepte der Sprache wird besonders auf die Anwendung von Java in der Webentwicklung eingegangen. Folgend wird das Webframework Play vorgestellt, welches, wie bereits erwähnt, zur direkten Entwicklung der Webanwendung verwendet wird. Da die Webanwendung den Anspruch hat ihre Benutzeroberflächen stets einheitlich und responsiv¹ zu präsentieren, was mit einem etablierten Framework komfortabler zu realisieren ist, wird das Front-End Framework² UIKit verwendet und unter 2.3 näher beschrieben.

Aufgrund des Ansatzes diese Entwicklung unter Aspekten der Softwareversionierung umzusetzen, wird abschließend die verwendete Software zur Versionsverwaltung kurz erläutert.

¹ Responsives Webdesign ist ein Paradigma, unter dem Designs auf Eigenschaften verschiedener Endgeräte reagieren und dadurch die Benutzerfreundlichkeit gewährleisten (siehe selfhtml.org [45, Artikel]).

² Front-End Frameworks sind Sammlungen von Gestaltungselementen und Hilfsmitteln für ein standardisiertes Design von Benutzeroberflächen (siehe t3n.de [48, Artikel]).

2.1 Die Programmiersprache Java im Web

2.1.1 Grundlegende Eigenschaften und Konzepte

Java ist eine objektorientierte Programmiersprache, welche unter dem Namen »The Green Project« vom Unternehmen Sun Microsystems entwickelt wurde. Zu den Hauptentwicklern zählten unter anderem Patrick Naughton, Mike Sheridan, James Gosling und Bill Joy³. Bei der Entwicklung von Java war eines der wichtigsten Hauptziele das Erreichen einer Plattformunabhängigkeit in Bezug auf die Ausführung von Programmen. Aufgrund dessen wird der Java Quelltext zunächst in einen sogenannten Java-Bytecode unter Verwendung des Java-Compilers kompiliert:

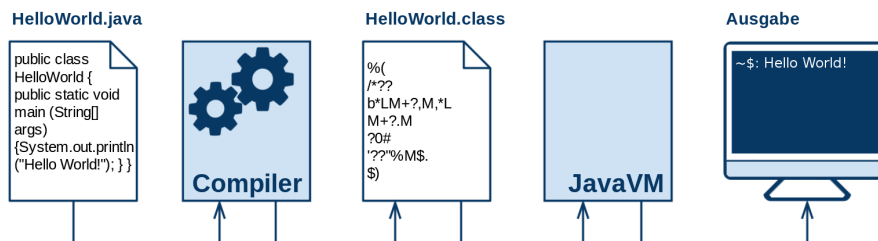


Abbildung 2.1: Java-Kompilierungsprozess, in Anlehnung an Quelle [24]

Dieser Bytecode wird auf dem System im Gegensatz zu vergleichsweise nativen Lösungen nicht direkt von der vorhandenen Hardware ausgeführt, sondern in einer Java-Laufzeitumgebung (eng. Java Runtime Environment) ([JRE](#)) erst zur Laufzeit in den Maschinencode übersetzt und danach ausgeführt. Diese Verfahrensweise, das Programm im Bytecode vorzuhalten, ermöglicht die Plattformunabhängigkeit. Notwendig ist zu jedem Zeitpunkt jedoch eine vorhandene [JRE](#). Dadurch soll gewährleistet werden, dass ein Programm unabhängig von der vorliegenden Rechnerarchitektur der Zielpattform ausgeführt werden kann.

³ siehe Buyya, Selvi, u. Chu [6, S.34]

2 Grundlegende Software und Konzepte

Für die Verfügbarkeit von JREs für verschiedene Systeme sind die unterschiedlichen Hersteller verantwortlich. Zum Beispiel bietet das Unternehmen Oracle, welches seit 2010 Sun Microsystems als Tochterunternehmen hat, JREs für die Systeme Linux, OSX, Windows und Solaris an. Wenn andere Betriebssysteme zum Einsatz kommen sollen, sorgen meist die jeweiligen Hersteller für eine Zertifizierung ihrer eigens entwickelten JREs. Dadurch ist es möglich, dass in Java geschriebene Software auch in Geräten, wie zum Beispiel einer großen Steuerungsanlage oder einer handelsüblichen Kaffeemaschine, zum Einsatz kommen kann.

Neben der Plattformunabhängigkeit steht bei der Entwicklung von Java auch das Programmierparadigma der Objektorientierte Programmierung (OOP) im Vordergrund. Der Grundgedanke der OOP besteht darin, Daten und Methoden in Objekten abzubilden und dadurch sinnvoll zu kapseln. Die Definition eines Objekts wird durch sogenannte Klassen geregelt. Der Vorteil dieser Programmiermethode liegt in den Konzepten der Kapselung und Vererbung. Das Konzept der Vererbung wird für den Aufbau neuer Klassen auf Basis bereits vorhandener Klassen genutzt, wobei die Beziehung zwischen Eltern- und Kindklassen dauerhaft ist. Ein weiterer Vorteil ist die mit der Abstraktion entstehende Modularität einer Software, welche sich in Folgeprojekten, durch Wiederverwendung von einzelnen Modulen als sehr effizient erweist (siehe Abbildung 2.2).

Java ist im Vergleich zu Scala⁴ nicht vollständig objektorientiert, da die Grunddatentypen⁵ keine Objekte sind. Diese können jedoch bei Bedarf via Autoboxing⁶ in die entsprechenden Objekte umgewandelt werden (ab Java 1.5).

⁴ siehe beuth-hochschule.de [4, PDF Präsentation]

⁵ Grunddatentypen in Java sind »boolean«, »char«, »byte«, »short«, »int«, »long«, »float« und »double«.

⁶ siehe oracle.com [34, S.34]

2 Grundlegende Software und Konzepte

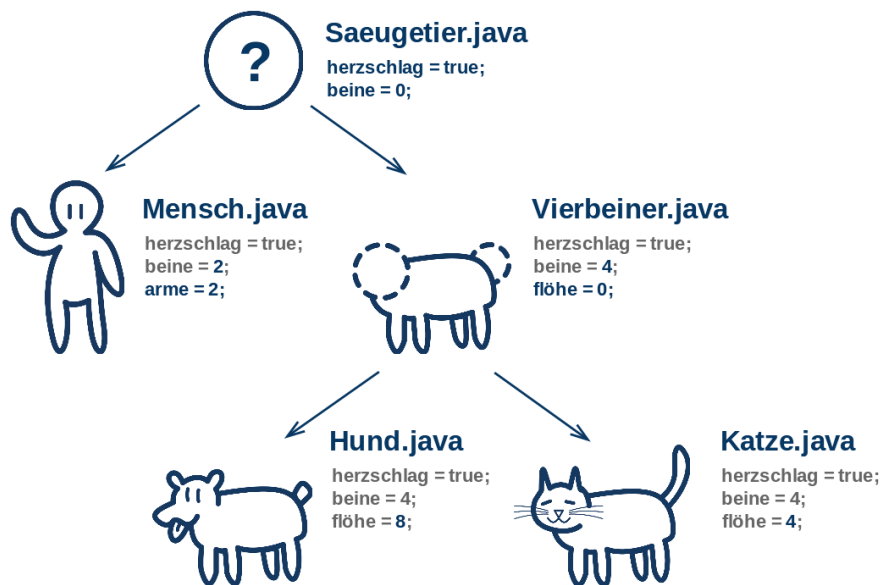


Abbildung 2.2: Klassenvererbung, in Anlehnung an Quelle [8]

Da auch im Hinblick auf das unter 2.2 vorgestellte Play Framework die Annotationen bei Java eine sehr bedeutende Rolle spielen, muss im Folgenden auf diese näher eingegangen werden. Ab Java 5 verfügt die Sprache über Annotationen, welche die Notation von Metadaten erlauben. Diese ermöglichen es die Sprache in einem begrenzten Rahmen zu erweitern. Durch die Verwendung spezieller Schlüsselwörter (Annotationen) kann Einfluss auf den Code genommen werden. Das ist zum Beispiel von Vorteil, um beim späteren Kompilieren bestimmte Codeblöcke automatisch erzeugen zu lassen. Eine beispielhafte Annotation ist im Listing 2.1 zu sehen.

```
1 @Entity
2 public class Saeugetier {}
```

Listing 2.1: Annotation in Java

Für die Softwareentwicklung mit Java stehen viele Integrierte Entwicklungsumgebungen (eng. Integrated Development Environments) (IDEs) zur Verfügung.

2 Grundlegende Software und Konzepte

Zu den bekanntesten IDEs gehören IBM Visual Age for Java, Metrowerks Codewarrior, WebGain Visual Café und Oracle JDeveloper ⁷. Nicht zu vergessen sind die beiden etablierten Open Source Anwendungen: das von der Eclipse Foundation bereitgestellte Eclipse und das von Sun entwickelte NetBeans. Für das OSX System gibt es ab der Version 10.3 die IDE Xcode, welche jedoch ihren Schwerpunkt auf die Entwicklung von C-, C++- und Swift- Software setzt.

Eine für Ausbildungszwecke angepasste IDE ist BlueJ. BlueJ ist in der Lage im Code vorkommende Beziehungen zwischen Klassen und Methoden grafisch unter Verwendung von Klassendiagrammen dynamisch darzustellen⁸, was bei vielen IDEs in Form von Plugins nachinstalliert werden kann.

2.1.2 Java in der Webentwicklung

Eine Webanwendung ist eine Anwendung, die nach dem Client-Server-Modell arbeitet. Das bedeutet, dass man bei einer Webanwendung immer von einem verteilten System ausgehen muss, bei dem die Geschäftslogik und Persistenz der Anwendung getrennt von der Benutzeroberfläche bzw. Bedienung ist. Die eben erwähnte Geschäftslogik wird in der Regel auf einen Webserver ausgelagert und dort implementiert. Ausschließlich die Ergebnisse der Anwendung werden zur Ausgabe an den Client übermittelt. Gesteuert bzw. angesprochen wird die Anwendung über einen Webbrowser (clientseitig), welcher die Kommunikation zum Webserver und letztlich zur Anwendung regelt. Der Webbrowser ist zudem für die Darstellung der Benutzeroberfläche der Webanwendung zuständig. Diese wird selbstverständlich auch vom Webserver geliefert. Wenn die Benutzeroberfläche eine sehr einfache Geschäftslogik verwendet, kann diese in einer dem Browser interpretierbaren Skriptsprache, wie zum Beispiel Javascript implementiert und bereitgestellt werden. Damit sind jedoch ausschließlich Manipulationen der Benutzeroberfläche (Graphical User Interface (GUI)) gemeint,

⁷ siehe teialehrbuch.de [49, Artikel]

⁸ siehe bluej.org [5, Offizielle Webseite]

2 Grundlegende Software und Konzepte

welche keine größeren Rechenleistungen vom Clientsystem benötigen. Siehe folgende Abbildung:

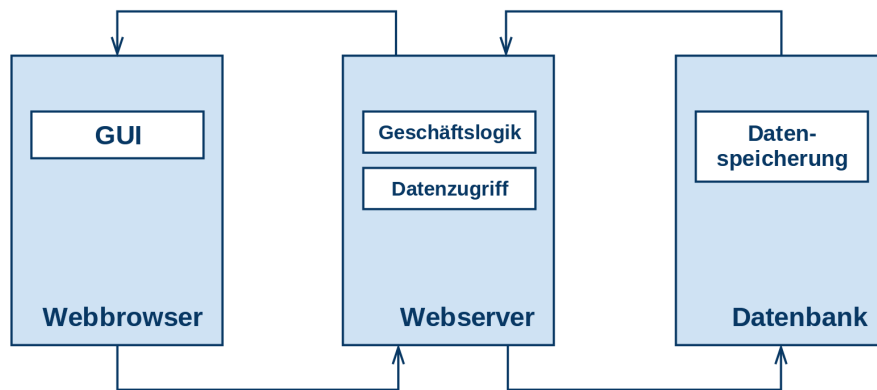


Abbildung 2.3: Webanwendung-Prozesse, in Anlehnung an Quelle [30]

Webanwendungen in Java besitzen demnach eine gewisse Plattformunabhängigkeit allein durch die Implementierung eines verteilten Systems. Dafür spricht, dass für die Benutzung einer Webanwendung aus Clientsicht ausschließlich ein Webbrowser notwendig ist, welcher in der Lage ist, die Benutzeroberfläche korrekt darzustellen. Dieser Webbrowser muss jedoch verschiedene Webstandards, wie zum Beispiel Hypertext Markup Language ([HTML](#)), Cascading Style Sheets ([CSS](#)) oder Javascript, unterstützen. Dies ist jedoch stets abhängig von den Anforderungen der jeweiligen Webanwendung.

Wenn die Geschäftslogik keine große Komplexität aufweist, kann sie problemlos ausgelagert und vom Client (Webbrowser) verarbeitet werden. Zur Implementierung dieser ausgelagerten Geschäftslogik wird meist eine Skriptsprache verwendet, welche vom Client (Webbrowser) interpretiert werden kann (zum Beispiel: Javascript). Die Auslagerung der Geschäftslogik zum Client eignet sich besonders bei Validierungsmethoden für Eingabedaten. Vor der Übermittlung der Daten an den Webserver, können diese bereits vom Webbrowser validiert werden. Diese Verfahrensweise spart einige Anfrage/Antwort-Prozesse, was

2 Grundlegende Software und Konzepte

sich positiv auf die Leistung einer Webanwendung auswirkt, da der Webbrowser des Clients insgesamt meist schneller mit solch kleinen Operationen, als der Webserver mit zuzüglichem Kommunikationsaufwand ist.

Für die Verwendung von Java in der Webentwicklung benötigt man Schnittstellen, welche es erlauben, die Anwendung über die im Web typischen Protokolle anzusprechen bzw. deren Antworten zu erzeugen. Die Schnittstellen sind hier Servlets⁹. Diese sind Java-Klassen, welche wie ein Webserver-Anfragen vom Client entgegennehmen und diesem auch wieder antworten können. Die Antworten werden hierbei dynamisch erzeugt und müssen dem Webserver nicht im Vorfeld bereit stehen (zum Beispiel in Form eines statischen [HTML](#)-Dokuments). Die Möglichkeit, Inhalte dynamisch erzeugen zu können, gibt es bereits mit dem Common Gateway Interface ([CGI](#))¹⁰, dessen Anfänge bis auf das Jahr 1993 zurückgehen. Somit kann man sagen, dass Servlets eine Art Weiterentwicklung von [CGI](#) darstellen. Eine Java-Servlet-Klasse implementiert meist die beiden Methoden »doGet()« und »doPost()« zum Verarbeiten der beiden wichtigsten Hypertext Transfer Protocol ([HTTP](#))-Methoden GET und POST. Alternativ kann auch nur die Methode »service()« überschrieben werden, welche automatisch auf die richtige do-Methode weiterleitet¹¹. Ein Servlet muss auf einem Webserver registriert werden, damit dieser das Routing richtig auflösen kann. Servlets benötigen einen Container, der alle Servlets verwalten kann. Dieser kann entweder in einem Webserver eingebettet sein oder in einem Applikationsserver. Ein Servlet-Container regelt nicht nur die Weiterleitung aller Anfragen zum jeweiligen Servlet, sondern verwaltet auch den Lebenszyklus eines Servlets. Ein einfach formulierter Prozessablauf¹² sieht wie folgt aus:

1. Ein Client sendet eine [HTTP](#)-Anfrage an einen Webserver.
2. Der Webserver registriert, dass das Ziel dieser Anfrage ein Servlet ist und

⁹ Servlet ist ein Kofferwort, welches sich aus den Begriffen Server und Applet zusammensetzt.

¹⁰ siehe [uni-mannheim.de](#) [[55](#), Artikel]

¹¹ siehe Ullenboom [[52](#), Kapitel 23.12 - Servlets]

¹² siehe Ullenboom [[52](#), Kapitel 23.2.1 - Servlet-Container]

2 Grundlegende Software und Konzepte

reicht die Anfrage an den Servlet-Container weiter.

3. Der Servlet-Container erzeugt eine Instanz von genau dem Servlet welches der Benutzer nutzen wollte, falls noch keine erzeugt wurde.
4. Letztlich übergibt der Servlet-Container sämtliche anfragebezogenen Daten dem richtigen Servlet.

Dadurch, dass ein Servlet-Container eine Sammlung an Servlets verwaltet und manche Servlet-Container mehrere Requests gleichzeitig abwickeln können, wird das Prinzip von Multi-Threading genutzt. Anders ausgedrückt, können mehrere Servlets parallel arbeiten, was einen großen Vorteil für die Leistung einer Anwendung mit sich bringt. Als Entwickler muss man dann jedoch auf Race Condition strikt achten.

Servlets berücksichtigen häufig Parameter und aktuelle Sitzungsdaten, um zum Beispiel eine personalisierte Antwort zu erzeugen, Daten auf dem Server zu verändern oder zu speichern.

Java-Servlets wurden von Sun Microsystems 1997 veröffentlicht. Bevor erste Frameworks auf der Basis von Servlets entstanden, übernahmen Servlets sowohl die Präsentation als auch die Steuerung von Java-Webanwendungen. Das führte zu sehr schwer wartbaren Anwendungen und war zudem auch sehr unproduktiv. Eine Lösung für dieses Problem wurde mit JavaServer Pages ([JSP](#)) im Jahr 1999 präsentiert. Mit [JSP](#) galt es, Steuerung und Präsentation voneinander zu trennen. Für die Steuerung (Controller) der Webanwendung sind ausschließlich die Servlets und für die Präsentation (Views) die [JSPs](#) verantwortlich. Nimmt man dann noch Klassen zur Datenhaltung und zusätzlicher Geschäftslogik hinzu, hat man eine Anwendung nach dem Entwurfsmuster Model View Controller ([MVC](#)) implementiert¹³. [MVC](#) ist ein Muster zur Entwicklung von Software, die aus drei Schichten besteht: dem Datenmodell (Model), der Präsentation (View) und der Steuerung (Controller):

¹³ siehe computerwoche.de [7, Artikel]

2 Grundlegende Software und Konzepte

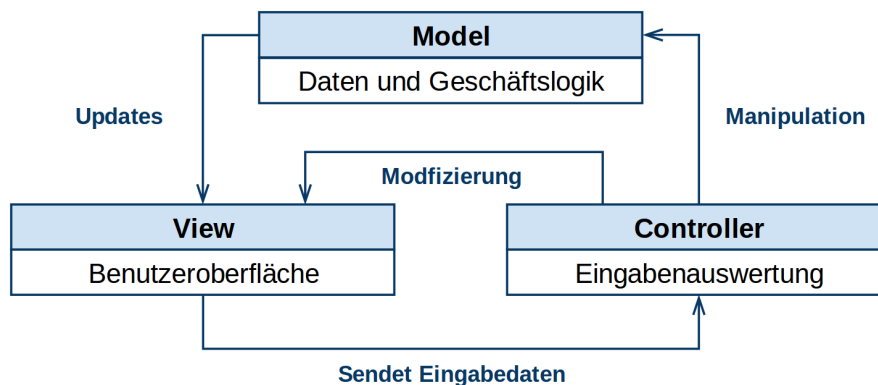


Abbildung 2.4: Modell-View-Controller, in Anlehnung an Quelle [31]

Das Ziel dieses Entwurfsmusters ist die Software sehr modular und flexibel aufzubauen, sodass eine spätere Änderung oder Erweiterung keinen großen Aufwand erfordert. Auch die Wiederverwendbarkeit einer oder zweier Schichten ist möglich. So können zwei verschiedene Programme zum Beispiel das gleiche Datenmodell nutzen, jedoch eine andere Steuerung sowie Präsentation anbieten. Hierfür müssen dann nur der Controller und die View neu implementiert werden. Das **MVC**-Modell wird anhand der Struktur von dem, unter Kapitel 2.2 beschriebenen Play Framework näher erläutert.

Eine Webanwendung auf der Basis ausschließlich von Servlets und **JSP** zu entwickeln, ist für größere Projekte sehr aufwendig und umständlich. Um Java in der Webentwicklung komfortabler zu gestalten sind bis dato viele Java-Web-Frameworks entstanden.

2.2 Das Webframework Play

2.2.1 Grundlagen und Struktur

Play ist ein Open Source Framework, für die Entwicklung von Webanwendungen mit Java und Scala. Das Framework folgt zwar der klassischen MVC-Architektur, bringt jedoch einige innovative Funktionalität mit. Zum Beispiel wird durch Funktionen wie Hot-Code-Reload¹⁴ oder das sofortige Ausgeben von Error-Logs¹⁵ im Browser ein hoher Komfort bei der Entwicklung garantiert.

Zu den proprietären Unternehmen, welche aktiv an der Entwicklung von Play beteiligt sind, zählen Lightbend (ehemals Typesafe¹⁶) und Zengularity¹⁷. Darüber hinaus wird das Framework von einer aktiven Open Source Community täglich verbessert und weiterentwickelt¹⁸.

Bei der Entwicklung mit Play erfolgt vieles durch die Benutzung der Kommandozeile, so auch das Initialisieren eines neuen Projekts mit: »activator new projektname play-java«. Das Programm Activator, welches von Play mitgeliefert wird, erstellt ein vollfunktionsfähiges Programmgerüst, das als Ausgang zur Entwicklung einer Webanwendung genutzt werden kann. Der Parameter »play-java« informiert über das bevorzugte Programmgerüst. Wenn die zu entwickelnde Webanwendung in Scala implementiert werden soll, bietet es sich an, gleich zu Beginn »play-scala« zu übergeben und sich somit ein Programmgerüst, welches in Scala geschrieben ist, zu erzeugen. Dieses neu initialisierte Projekt lässt sich bereits kompilieren und ausführen. Play besitzt mit dem Client-Server-Framework Netty (JBoss Netty) eine integrierte Software, welche

¹⁴ Bei verändertem Code wird auf eine [HTTP](#)-Anfrage das Projekt neu kompiliert und Änderungen werden sofort sichtbar.

¹⁵ Play stellt, unter Verwendung des Entwickler Modus Exceptions im Browser dar.

¹⁶ siehe [lightbend.com](#) [28, Artikel]

¹⁷ siehe [playframework.com](#) [37, Offizielle Webseite]

¹⁸ siehe [github.com](#) [21, Artikel]

2 Grundlegende Software und Konzepte

den Einsatz einer zusätzlichen Webserver-Software ersetzen soll. Demnach ist das Ausführen der Webanwendung möglich, ohne einen größeren Aufwand an Deployment-Arbeit¹⁹. Jedoch empfiehlt es sich im Produktivbetrieb einen etablierten Webserver mit entsprechenden Containern zu verwenden und das jeweilige Projekt als Web Application Archive (WAR) zu verpacken. Mit diesem WAR-Archiv können dann alle Webserver arbeiten, die einen Container für WAR-Archive bereitstellen.

Mit dem Befehl »activator run« wird das Projekt kompiliert und mit dem eben erwähnten Dienst »Netty« standardmäßig über den Port »9000« bereitgestellt. Während der Ausführung bzw. Bereitstellung der Webanwendung sind Veränderungen am Code möglich und sogar erwünscht. Diese Änderungen werden nach einer neuen HTTP-Anfrage (Webseite neuladen) sofort sichtbar bzw. es wird eine entsprechende Fehlermeldung in der Weboberfläche ausgegeben, falls die Änderung im Code nun einen Fehler (Java: Exception) liefert. Eine typische Exception und deren Fehlermeldung sind beispielhaft in der Abbildung 2.5 zu sehen.

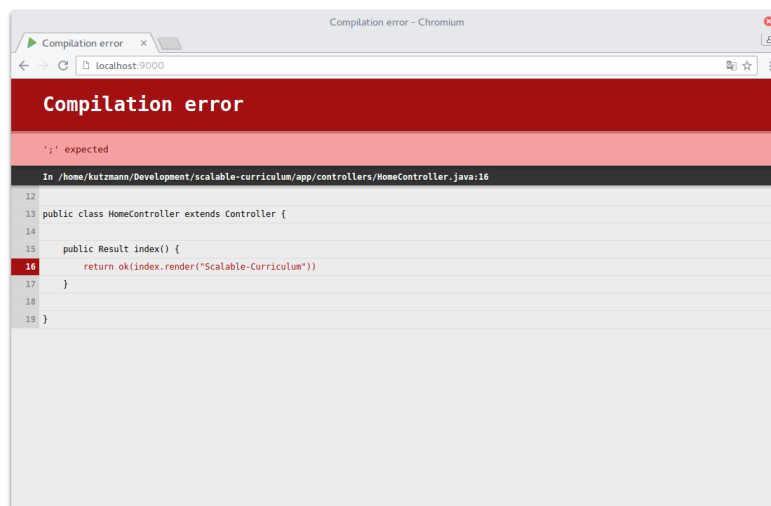


Abbildung 2.5: Hot-Code-Reload mit beispielhafter Fehlermeldung

¹⁹ Mit »Deployment« ist in diesem Zusammenhang der Installations- und Inbetriebnahme-prozess einer Webanwendung gemeint.

2 Grundlegende Software und Konzepte

Genau in dieser Hot-Code-Reload Funktion liegt ein großer Vorteil bei der Entwicklung mit Play gegenüber anderen Webframeworks. Wenn ein Fehler beim Kompilieren des Projekts auftritt, kann immer davon ausgegangen werden, dass in der Fehlermeldung sowohl die betroffene Datei als auch die fehlerhafte Zeile ausgewiesen ist.

Zu Beginn wurde Play als ein klassisches **MVC**-Framework bezeichnet. Dieses Modell trennt die Webanwendung in verschiedene Schichten bzw. Module: die Präsentationsschicht, die Kontrollschicht und die Datenschicht.

Als Model wird die Datenschicht einer Webanwendung bezeichnet. Sie beinhaltet die domänenspezifischen Darstellungen aller Informationen, mit der die Webanwendung arbeitet. Dazu zählen nicht nur die Informationen in Form von Datensätzen sondern auch Methoden, welche domainspezifische Algorithmen implementieren, werden in dieser Schicht gehalten. In der Regel verwenden die meisten Webanwendungen einen persistenten Speichermechanismus, wie zum Beispiel eine relationale Datenbank (**RDB**). Unter Model, ist jedoch nicht der Datenzugriff zu verstehen, sondern vielmehr die Datenabbildung in Form von Objekten. Oft kommen jedoch Object-Relational Mapping (**ORM**)-Frameworks zum Einsatz, welche als Hauptaufgabe das Ablegen und Verwalten von Objekten (im Sinne der **OOP**) in einer Datenbank haben.

Die Oberflächenschicht bzw. View ist für die Darstellung der Benutzeroberfläche einer Webanwendung (meist **GUI** genannt) verantwortlich. Sie stellt nicht nur die eben erwähnten Daten dar. Mit der Oberfläche werden dem Benutzer unter anderem auch Möglichkeiten zur Manipulation von Daten bereitgestellt. Hierbei kann ein einziges Model über mehrere Ansichten für die Bereitstellung verschiedenster Methoden verfügen. Die Darstellung erfolgt im Normalfall in einem Web-Format, wie zum Beispiel **HTML**, JavaScript Object Notation (**JSON**), oder Extensible Markup Language (**XML**). Möglich sind auch binäre Ausgaben, wie zum Beispiel Grafiken.

2 Grundlegende Software und Konzepte

Die Kontrollschicht oder Controller dient als Mittelschicht zwischen der Benutzeroberfläche und der Datenschicht, welche Aktionen des Benutzers auswertet und verarbeitet. So kann die Kontrollschicht zum Beispiel Manipulationsmethoden der Datenschicht aufrufen. In einer Webanwendung sind die Aktionen in der Regel [HTTP](#)-Anfragen. Diese Anfragen werden von den jeweiligen Controllern in die einzelnen Bestandteile wie Abfrage-String-Parameter, Request-Header etc. zerlegt und ausgewertet²⁰.

In Play sind die drei Schichten in jeweils separate Packages²¹ getrennt. Dies verdeutlicht auch die Klassen- bzw. Dateienorganisation innerhalb des Verzeichnisses »app/«:

```
play-project/ .....Wurzelverzeichnis des Projekts
├── app/
│   ├── classes/ .....Extraklassen (siehe Anhang B.5)
│   ├── controllers/ .....Package: controllers
│   ├── models/ .....Package: models
│   ├── views/ .....Package: views
│   └── [...]
└── [...]
```

Abbildung 2.6: Verzeichnisbaum einer MVC-Struktur

[app/models](#)

Im Package »models« sind alle Daten in Klassen abgebildet, mit denen die Webanwendung arbeitet. Für deren Implementierung sind die Prinzipien der [OOP](#) maßgebend. Ein Model sollte stets alle Attribute mit Setter- und Gettermethoden bereitstellen sowie die Ausweisung der Datensatzart durch Annotationen mit dem Schlüsselzeichen »@« (siehe Listing 2.2 Zeile 7 und 9). Für die Verwaltung

²⁰ siehe playframework.com [39, Dokumentation - The Main Concepts]

²¹ Ein Package (Paket) enthält eine oder mehrere Klassen, die sich einen Geltungsbereich (Namespace) für Klassen teilen (siehe uni-mannheim.de [54, Artikel]).

2 Grundlegende Software und Konzepte

der Daten in einer Datenbank bringt Play zum Beispiel das [ORM-Framework Ebean](#)²² mit. Mit diesem Framework stehen viele nützliche Methoden zur Verwaltung von Objekten (zum Beispiel: Löschen, Anlegen, Updaten) automatisch zur Verfügung. Die Voraussetzung ist, dass hierbei eine Modelklasse von »play.db.ebean.Model« erbt. Auch das automatische Anlegen von notwendigen Tabellen, zum Beispiel Mapping-Tabellen²³ wird durch das entsprechende [ORM-Framework](#) bereitgestellt. Bei der Entwicklung mit Play ist man nicht auf ein bestimmtes [ORM-Framework](#) eingeschränkt, sodass Projekte zum Beispiel genauso gut mit Hibernate, Slick²⁴ oder Java Persistence API ([JPA](#)) realisiert werden können.

```
1 package models;
2 [...]
3 @Entity
4 @Table(name = "users")
5 public class User extends Model {
6
7     @Id
8     private UUID id;
9     @Column(nullable = false, unique=true)
10    private String email;
11
12    //Setter
13    public void setEmail(String email){
14        this.email = email;
15    }
16 [...]
```

Listing 2.2: Auszug eines beispielhaften User-Models

app/views

Eine effiziente Lösung zur Umsetzung von vielen unterschiedlichen, aber dennoch visuell einheitlichen grafischen Benutzeroberflächen stellt eine Template-

²² siehe ebean-orm.github.io [10, Offizielle Webseite]

²³ Mapping Tabellen dienen der Speicherung von Beziehungen zwischen einzelnen Models.

²⁴ Slick ist eine moderne SQL-Statement- und Zugriffsbibliothek für Scala.

2 Grundlegende Software und Konzepte

Engine dar. Play bringt eine eigene Template-Engine mit, welche in Scala geschrieben ist und deren Templates auch in Scala geschrieben werden. Ein typischer Aufbau einer solchen Template-Datei ist im Listing 2.3 zu sehen.

```
1 @(message: String)
2
3 <html>
4   <head>
5     <title>Beispiel Template</title>
6   </head>
7   <body>
8     <h1>@message</h1>
9   </body>
10 </html>
```

Listing 2.3: Einfaches Scala Template

Einem Template können selbstverständlich Parameter übergeben werden. Diese müssen jedoch eindeutig bezeichnet werden, damit sie innerhalb des Templates bekannt sind (siehe hierzu Listing 2.3 Zeile 1). In Zeile 8 ist zu erkennen, wie Zugriffe auf diese Parameter realisiert werden. Einem Template können sogar ganze Objekt-Instanzen übergeben werden, sodass der Code auch wie folgt aussehen kann (unter der Voraussetzung, dass »getName()« einen String zurück gibt):

```
1 @(user: models.User)
2 [...]
3   <h1>@user.getName() </h1>
4 [...]
```

Listing 2.4: Objekte innerhalb von Scala Templates

app/controllers

Unter dem Verzeichnis »app/controllers« befinden sich alle Controller der Webanwendung. Ein Controller ist eine Java-Klasse, welche »public static«-Methoden enthält und von »play.mvc.Controller« erbt. Diese Methoden sind

2 Grundlegende Software und Konzepte

sogenannte Actions. Actions sind als Einstiegspunkte gedacht und verarbeiten die Daten der [HTTP](#)-Anfragen. Sie führen je nach Zweck weitere Methoden aus und regeln demnach den Aktionsfluss der Webanwendung. In der Regel sendet eine Aktion unter Verwendung einer bestimmten View eine Antwort an den Benutzer:

```
1 package controllers;
2 [...]
3 public class HomeController extends Controller {
4     public Result index() {
5         return ok(index.render("hello world"))
6     }
7 }
```

Listing 2.5: Beispielhafter Controller

Die Regeln, welcher Controller mit welcher Aktion auf eine bestimmte Route²⁵ antwortet, werden in der Konfigurationsdatei »conf/routes« beschrieben. Um den in Listing 2.5 gezeigten Controller über »localhost:9000/home« aufrufen zu können, muss die Konfigurationsdatei die Zeile 1 aus dem Listing 2.6 enthalten.

```
1 GET      /home                                controllers.HomeController.index
```

Listing 2.6: Notwendige Routing-Konfiguration

Da Controller prozessorientiert sind, wird für deren Implementierung meist der Stil der prozeduralen Programmierung bevorzugt.

²⁵ Eine Route ist in diesem Kontext eine aufrufbare Uniform Resource Locator ([URL](#)).

2.3 Das Front-End Framework UIKit

2.3.1 Grundlagen und Entstehung

Im folgenden Abschnitt wird *ausschließlich* die Entwicklung der Benutzeroberflächen von Webanwendungen (Views) beschrieben.

UIKit²⁶ ist ein schlankes und modulares Front-End Framework für die schnelle und effiziente Entwicklung von Weboberflächen, welches am 19. Juli 2013²⁷ in der ersten Version vom hamburger Unternehmen YOOtheme GmbH²⁸ veröffentlicht wurde. Das Framework liefert eine umfassende Sammlung an [HTML](#)-, [CSS](#)- und Javascript-Komponenten, deren Verwendung, Bearbeitung und Erweiterung benutzerfreundlich ist. Für die verwendeten [CSS](#)-Komponenten wird auf die LESS Stylesheet-Sprache ([LESS](#))²⁹ gesetzt, was eine schnelle und komfortable Bearbeitung und Erweiterung gewährleisten soll. Die Lizenz von UIKit erlaubt eine freie Verwendung (siehe Open Source Initiative [[32](#), MIT-Lizenz]) und wird aktiv von der YOOtheme GmbH, aber auch freiwilligen Entwicklern weiterentwickelt.

2.3.2 Bestandteile des Frameworks

UIKit ist für das Entwickeln von Weboberflächen mit den Techniken [HTML](#), [CSS](#) und Javascript entworfen worden. Aufbauend auf diesen drei Techniken liefert UIKit eine Sammlung an Hilfsmitteln und Vorlagen für die Gestaltung von Weboberflächen. Das Framework ist in sechs Bereiche gegliedert³⁰, von Standard-[HTML](#) Elementen, wie zum Beispiel Überschriften oder Codeblöcken,

²⁶ siehe [getuikit.com](#) [[14](#), Offizielle Webseite]

²⁷ siehe [github.com](#) [[19](#), UIKit - Releases]

²⁸ siehe [yootheme.com](#) [[56](#), Offizielle Webseite]

²⁹ siehe [lesscss.org](#) [[25](#), Offizielle Webseite]

³⁰ siehe [getuikit.com](#) [[16](#), Dokumentation - Base]

2 Grundlegende Software und Konzepte

über Navigationselemente bis hin zu Javascriptelementen, wie zum Beispiel einer Dropdown-Liste oder einem smarten Modaldialog.

DEFAULTS

DEFAULTS sind standardtisierte Stylesheets, um Cross-Browser-Unterschieden entgegen zu wirken und ein minimales grundlegendes Styling zu erzeugen.

LAYOUT

Das LAYOUT beinhaltet das Grid-System und modulare Gestaltungsbestandteile wie Artikel, Kommentare und diverse Hilfsklassen, wie zum Beispiel die Flex-Klasse, welche es erlaubt dynamische Größen bzw. Spannweiten zu vergeben, oder die Cover-Klasse, die es ermöglicht Bilder und Videos effektiv in einen Container einzubetten.

NAVIGATIONS

Unter dem Gliederungspunkt NAVIGATIONS sind alle Stylesheets und Hilfsklassen zusammengefasst, die für die korrekte Erstellung einer Navigation notwendig sind. UIKit bietet unterschiedliche Arten von Navigationsleisten, Seiten-Navigtionen und Breadcrumbs³¹ an.

ELEMENTS

ELEMENTS fasst alle Stylings für [HTML](#)-Elemente zusammen, welche nicht mit Multiklassifizierung³² definiert werden können. Diese Elemente besitzen

³¹ Unter Brotkrümelnavigation (englisch breadcrumb navigation) ist ein Entwurfsmuster für die Gestaltung grafischer Benutzeroberflächen zu verstehen, welches die Navigation in einer Textzeile abbildet.

³² Multiklassifizierung beschreibt hier die Verfahrensweise einem [HTML](#)-Element mehrere [CSS](#)-Klassen zuzuweisen.

2 Grundlegende Software und Konzepte

eigene Klassendefinitionen und können zum Beispiel nicht noch zusätzlich der Klasse Flex angehören. Typische ELEMENTS sind Tables und Forms.

COMMON

Zu COMMONs zählen Buttons, Icons, Alerts, Thumbnails, Overlays, Animationen und weitere Elemente, die für sich stehen, jedoch im Vergleich zu Tables und Forms von anderen Klassen erben können.

JAVASCRIPT

Die Javascript-Komponenten von UIKit bieten unter anderem Dropdowns, modale Dialogfenster, Content-Switcher und eine ausgereifte Scroll-To-Funktion.

2.3.3 Anwendungsbeispiel

Im Folgenden wird die Anwendung von UIKit an einem Beispiel beschrieben. Zu Beginn muss das Framework in dem Projekt verfügbar gemacht werden. In dem entsprechenden [HTML](#)-Dokument bzw. in der verwendeten Template-Engine müssen alle notwendigen Dateien eingebunden werden. Im Listing 2.7 ist zu erkennen, wie eine minimale Einbindung im Element »head« aussieht.

```
1 <head>
2   <title></title>
3   <link rel="stylesheet" href="uikit.min.css" />
4   <script src="jquery.js"></script>
5   <script src="uikit.min.js"></script>
6 </head>
```

Listing 2.7: Einbindung von UIKit

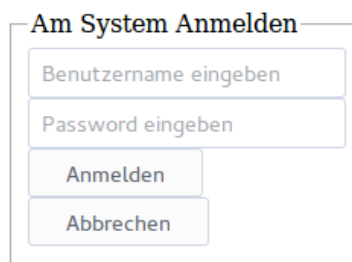
Mit diesem Setup greifen bereits die grundlegenden Stylesheets, welche zuvor als DEFAULTS beschrieben wurden und nehmen direkten Einfluss auf die Gestaltung der Weboberfläche. Über die Verwendung von bestimmten [CSS](#)-Klassen

2 Grundlegende Software und Konzepte

entsprechend der UIKit-Dokumentation lassen sich weitere [HTML](#)-Elemente anpassen oder bestimmte Layout-Komponenten erzeugen. Diese Klassifizierung von [HTML](#)-Elementen durch UIKit-[CSS](#)-Klassen beschreibt im Wesentlichen die gesamte Entwicklungsarbeit mit dem UIKit Framework.

```
1 <form>
2 <fieldset>
3   <legend>Am System Anmelden</legend>
4   <div><input type="text" placeholder="Benutzername eingeben"></div>
5   <div><input type="password" placeholder="Password eingeben"></div>
6   <div>
7     <button type="button">Anmelden</button>
8     <button type="button">Abbrechen</button>
9   </div>
10 </fieldset>
11 </form>
```

Listing 2.8: HTML-Form ohne UIKit



Am System Anmelden

Benutzername eingeben

Password eingeben

Anmelden

Abbrechen

Abbildung 2.7: HTML-Form ohne UIKit

Das Listing 2.8 zeigt einen Quelltext-Ausschnitt einer gewöhnlichen [HTML](#)-Form ohne jegliche Stylesheet-Klassen oder Ähnliches. Das Ergebnis, wie in [Abbildung 2.7](#)³³ zu sehen, ist eine Form ohne Modifikation des Styles via [CSS](#)-Regeln. Die [Abbildung 2.8](#) zeigt hingegen eine Form, welche durch UIKit-Klassen modifiziert wurde, jedoch im Grundaufbau mit der zuvor gezeigten

³³ Aufgerufen im Webbrowser Mozilla Firefox (v.47.0)

2 Grundlegende Software und Konzepte

übereinstimmt. Wie dem Listing 2.9 zu entnehmen ist, wurden einige [HTML](#)-Elemente gemäß der Dokumentation³⁴ von UIKit mit [CSS](#) klassifiziert. Als Voraussetzung gilt natürlich die Einbindung der UIKit-Dateien (siehe Listing 2.7) im »head« des [HTML](#)-Dokuments.

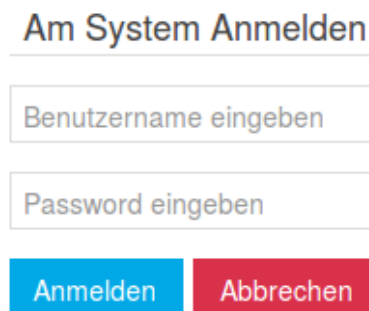


Abbildung 2.8: HTML-Form mit UIKit

```
1 <form class="uk-form">
2   <fieldset>
3     <legend>Am System Anmelden</legend>
4     <div class="uk-form-row"><input type="text" placeholder="..."></div>
5     <div class="uk-form-row"><input type="password" placeholder="..."></div>
6     <div class="uk-form-row">
7       <button class="uk-button uk-button-primary" type="button">Anmelden</button>
8       <button class="uk-button uk-button-danger" type="button">Abbrechen</button>
9     </div>
10  </fieldset>
11 </form>
```

Listing 2.9: HTML-Form mit UIKit

Als erstes wird die Form klassifiziert. Diese wurde als »uk-form« definiert, was grundlegende Layout-Modifizierungen festlegt. Die Elemente »fieldset« und »legend« werden durch [CSS](#)-Vererbung³⁵ beeinflusst, dass heißt für diese Elemente sind keine speziellen [CSS](#)-Klassifizierungen notwendig, solange diese

³⁴ siehe getuikit.com [17, Dokumentation - Core]

³⁵ siehe thestyleworks.de [50, Artikel]

2 Grundlegende Software und Konzepte

innerhalb des »uk-form«-Elements liegen. Durch die Klasse »uk-form-row« wird jeweils ein einheitlicher Block generiert, der als Zeile innerhalb des Form-Elements dargestellt wird. Bei den Input-Elementen »text« und »password« wirken wieder, die eben erwähnten [CSS](#)-Vererbungsregeln und modifizieren diese mithilfe von [CSS](#)-, aber auch Javascript-Klassen³⁶. Die Buttons »Anmelden« und »Abbrechen« werden zunächst identisch als »uk-button« klassifiziert. Der signifikante Unterschied zwischen den beiden Buttons besteht in der weiteren Klassifizierung und wird durch unterschiedliche Farben erkennbar (siehe Abbildung 2.8). In UIKit stehen dem Entwickler, für Buttons, Navigationen und Badges folgende vier globale Farbcodes zur Verfügung:

- primary background,
- success background,
- warning background und
- danger background.

UIKit besitzt eine große Menge an vordefinierten Einstellungen. Wenn der Entwickler bzw. Designer jedoch Eingriff in die Einstellungen nehmen und das Framework auf die eigenen Anforderungen anpassen möchte, sollte der Customizer³⁷ genutzt werden, welcher auf der offiziellen Webseite von UIKit zur Verfügung steht. Dieses Werkzeug bietet die Möglichkeit alle Einstellungen zu verändern und deren Auswirkung in Echtzeit nachzuvollziehen. Alternativ können sich Entwickler auch den [LESS](#)-Code herunterladen, diesen editieren und die notwendigen [CSS](#)-Dateien selbst kompilieren und einbinden.

³⁶ Javascript regelt die Darstellung von Effekten, wie zum Beispiel bei »onclick«- oder »onblur«-Events.

³⁷ siehe getuikit.com [15, Werkzeug]

2.4 Softwareversionierung mit Git

2.4.1 Entstehung

Git ist eine freie Software zur verteilten Versionierung von Dateien. Die Entwicklung und Implementierung von Git wurde im April 2005 von Linus Torvalds initiiert³⁸, da er mit der bis dato vorhandenen Open Source Versionierungssoftware nicht zufrieden war. Die starke Verbreitung und zahlreiche Anwendung spricht für den Erfolg von Git. Die meisten neuen Open Source Projekte nutzen Git als den Standard für die Software-Versionierung. Auch ältere Projekte sind mittlerweile zu Git migriert. Git ist nicht nur für Open Source Projekte, bei denen die Entwickler über die ganze Welt verteilt sind sinnvoll, sondern auch in kleinen Entwicklerkreisen bzw. bis hin zum Einzelprojekt.

2.4.2 Grundlagen

Dieser Abschnitt beschäftigt sich thematisch mit der Informations- und Versionsverwaltung von Git.

Der Hauptunterschied zwischen Git und anderen Versionskontrollsystemen besteht darin, wie Git Dateien betrachtet. Andere Versionskontrollsysteme, wie zum Beispiel Concurrent Versions System (CVS)³⁹, bilden die Versionen anhand der Änderungen an den Dateien ab. Diese Systeme verwalten demnach eine Menge an Dateien und deren Änderungen, welche über den Entwicklungszeitraum hinweg entstanden sind (siehe Abbildung 2.9).

³⁸ siehe Stachmann u. Preißel [47, Seite VII]

³⁹ siehe uni-bonn.de [53, PDF Artikel]

2 Grundlegende Software und Konzepte

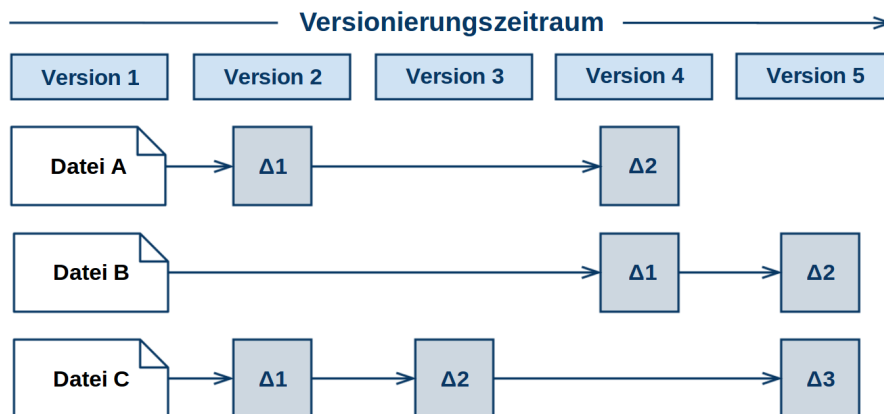


Abbildung 2.9: Versionierung anhand von Dateienänderung, in Anlehnung an Quelle [18]

Git betrachtet hingegen nicht die Änderung, sondern die spezielle Datei, welche sich verändert hat. Bei jedem Commit wird ein sogenannter Schnappschuss von dem gesamten Projekt und damit allen Dateien erzeugt. Schnappschuss bedeutet in diesem Fall, dass alle veränderten Dateien kopiert werden und zu allen unveränderten Dateien wird lediglich eine Verknüpfung zu der letzten Version gesetzt. Das Setzen einer Verknüpfung, bei unveränderten Dateien, hält das System sowohl effizient (Speicherbedarf), als auch leistungsstark (Update-Geschwindigkeit). Die Abbildung 2.10 soll zeigen, wie eine Versionierung allgemein mit Git funktioniert.

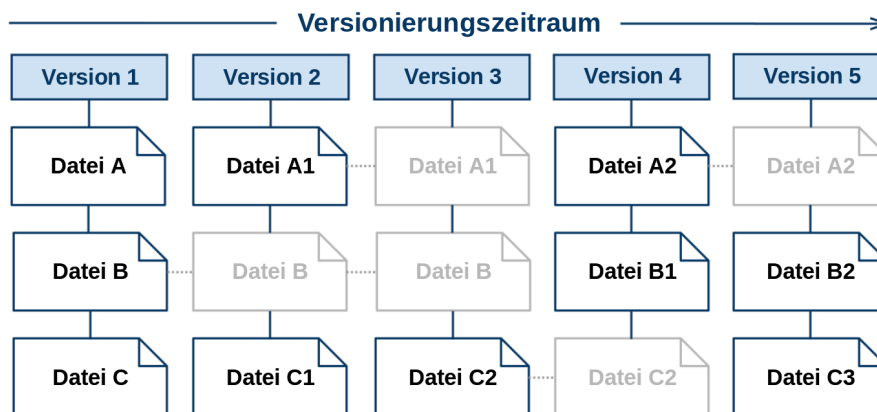


Abbildung 2.10: Versionierung anhand von Schnappschüssen, in Anlehnung an Quelle [29]

2 Grundlegende Software und Konzepte

Diese Art der Informationsverwaltung ist der markanteste Unterschied zwischen Git und anderen Versionskontrollsystemen.

Git ist wie oben beschrieben, eine freie Software zur verteilten Versionsverwaltung von Dateien. Das Konzept der verteilten Versionsverwaltung ist im direkten Vergleich zu dessen Pendant der zentralen Versionsverwaltung gut zu erklären. Eine zentrale Versionsverwaltung beschreibt das Arbeiten mit einem Repository⁴⁰, welches zentral verfügbar ist und auf dem alle Nutzer gleichermaßen zugreifen. Zudem besitzt jeder Nutzer ein lokales Arbeitsverzeichnis (Working Directory), indem er aktiv Änderungen an den Dateien vornehmen kann. Diese Änderungen werden in Folge des nächsten Commits an das Repository übermittelt und als neue Version registriert (siehe Abbildung 2.11).

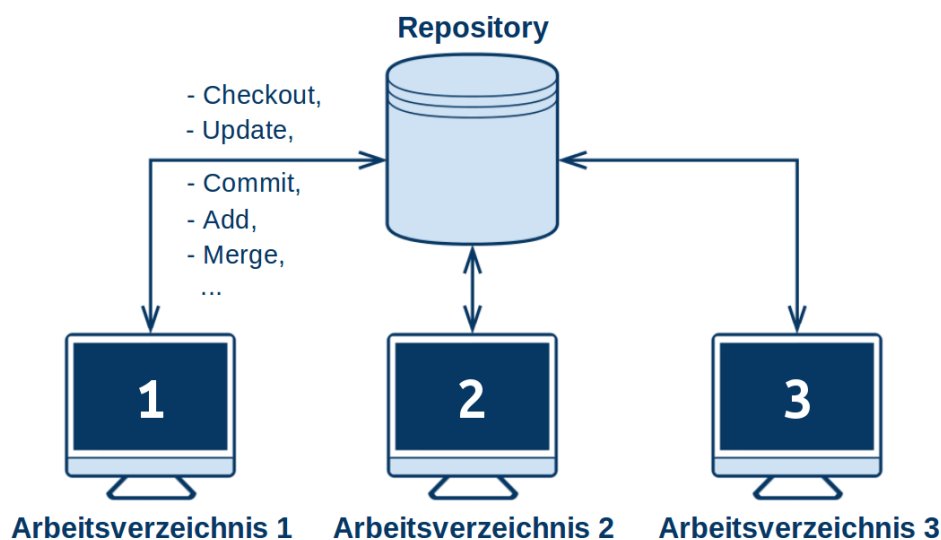


Abbildung 2.11: Zentrale Versionsverwaltung, in Anlehnung an Quelle [11]

Ein großer Nachteil der Entwicklung mit zentraler Versionsverwaltung ist, dass es *ausschließlich* ein Repository gibt und dieses auf einem zentralen Server liegt.

⁴⁰ Das Repository ist das (zentrale) Archiv, welches alle Revisionen sowie Dateien neben dazugehöriger Logdateien in Form einer Baumstruktur enthält (siehe fh-wedel.de [13, Artikel]).

2 Grundlegende Software und Konzepte

Folglich ist für jedes Commit eine Netzwerkverbindung zu dem Repository-Server notwendig, was insbesondere Entwickler *unterwegs* mit mobilen Geräten (zum Beispiel Notebooks) stark einschränkt. Weiterhin ist ein Commit über Netzwerk-Schnittstellen immer langsamer als ein lokaler Commit. Ein wichtiger Punkt ist auch, dass jede Revision sofort für alle anderen Entwickler sichtbar ist und es nicht die Möglichkeit gibt, experimentelle Revisionen zu erstellen, ohne dass diese gleich öffentlich sind.

Bei einer verteilten bzw. dezentralen Versionsverwaltung verfügt im Unterschied zur zentralen Versionsverwaltung jeder Nutzer über ein eigenes lokales Repository. In diesem Repository kann der Nutzer seine Änderungen versionieren (mittels Commits), ohne dass diese für andere Entwickler sichtbar sind (sozusagen privat). Diese Versionierung ist im Vergleich zur zentralen Versionsverwaltung schneller, da kein Netzwerkzugriff benötigt wird. Hinzu kommt, dass jeder Commit konfliktfrei ist und somit zum Beispiel keine Probleme bei einem Merge auftreten können. Der Austausch bzw. öffentliche Abgleich der Revisionen erfolgt dann über Pull-/Push-Befehle, welche das lokale mit dem entfernten Repository abgleichen. Optional kann auch ein gemeinsames Repository für den Austausch genutzt werden (siehe Abbildung 2.12).

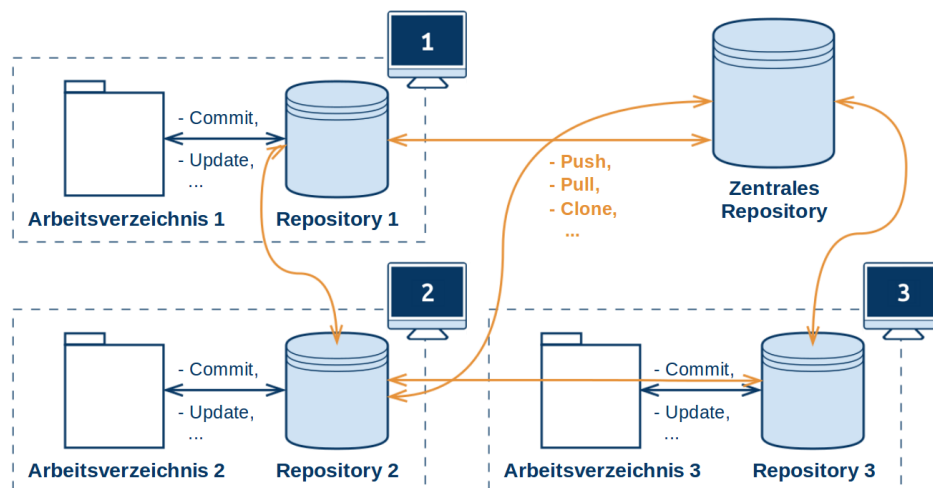


Abbildung 2.12: Verteilte Versionsverwaltung, in Anlehnung an Quelle [12]

Natürlich hat auch dieses Konzept der Versionsverwaltung einige Nachteile. Zum Beispiel erhöht sich der Speicherplatzbedarf aufgrund des lokalen Repositories wesentlich. Ein weiterer Aspekt ist, dass Revisionen nun überall erzeugt werden können und deshalb die Revisionsnummern globale Eindeutigkeit garantieren müssen. Demnach nimmt die Komplexität der Revisionsnummern zu, was den Zugriff auf bestimmte Revisionen erschwert. Aufgrund der Tatsache, dass mehrere Nutzer gleichzeitig und unabhängig voneinander Revisionsstrukturen lokal erzeugen können, kann die Gesamt-Revisionsstruktur sehr komplex werden.

2.4.3 Allgemeiner Arbeitsablauf

Einen typischen Arbeitsablauf bei der Entwicklung mit Git über Versionszweige (Branches) zeigt die Abbildung 2.13, sowie die dazugehörigen Handlungsanweisungen:

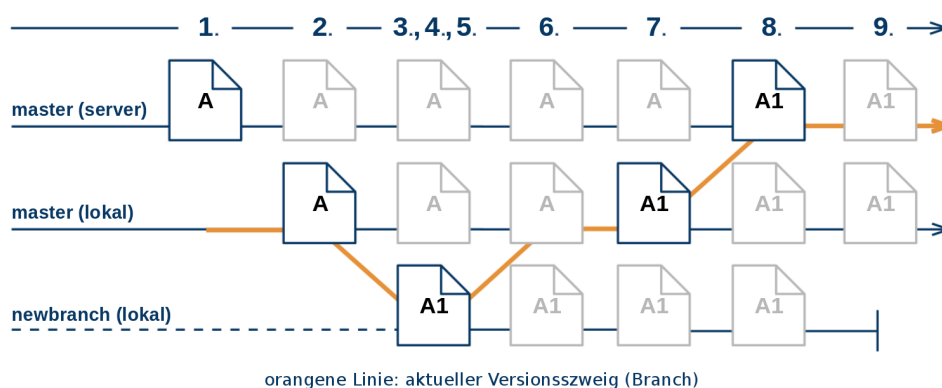


Abbildung 2.13: Braching Workflow

1. Auf den aktuellen Master-Branch wechseln:
git checkout master
2. Downloaden des aktuellen Master-Branchs vom Repository-Server:
git pull

2 Grundlegende Software und Konzepte

3. Einen neuen Branch für die anstehende Änderung (Feature, Bugfix oder ähnliches) anlegen:
git checkout -b newbranch master
4. Nachdem am Code Änderungen vorgenommen wurden, werden die geänderten oder neuen Dateien für ein Commit registriert:
git add »datei« oder alle Dateien mit einmal **git add - -all**
5. Committen:
git commit -m „Beschreibung der Revision“
6. Wieder in den Master-Branch zurückwechseln:
git checkout master
7. Und die beiden Branches zusammenführen:
git merge newbranch
8. Neue Revision des Master-Branches auf den Server laden mit:
git push
9. Abschließend den Branch »newbranch« löschen:
git branch -d newbranch

3 Theoretischer Entwurf der Webanwendung

Dieser Abschnitt behandelt den theoretischen Entwurf der Webanwendung. Hierfür werden zu Beginn die zentralen, funktionalen, sowie die nicht funktionalen Anforderungen an die zu entwickelnde Software ermittelt. Unter Verwendung der Ergebnisse dieser Anforderungsanalyse werden die funktionalen Anforderungen im nächsten Abschnitt nach Ihrer Funktionsweise definiert und unter Berücksichtigung des Gesamtkonzepts zueinander in Beziehung gesetzt und beschrieben.

Als Einstiegspunkt für die Implementierung der Play-Applikation dient eine Übersicht über alle der zu implementierenden Module in Form eines Komponenten-Diagramms. Das ist der Grund dafür, dass dieses Kapitel mit seinem letzten Abschnitt ein spezielles, auf den Anforderungen und funktionalen Komponenten basierendes Diagramm darstellt und dieses erklärt. Die anschließende Umsetzung mit dem Java Play Framework dieses Diagramms wird im folgenden Kapitel behandelt und bildet die Grundlage der Implementierung des Softwareentwurfs.

3.1 Zentrale Anforderungen an die Webanwendung

3.1.1 Funktionale Anforderungen

Ziel ist die Entwicklung einer Webanwendung, welche die folgenden Spezifikationen und Schnittstellen unter Berücksichtigung der zuvor festgelegten Rahmenbedingungen nachweist:

Übungen als zentrale Objekte der Anwendung

Die zentralen Objekte der zu entwickelnden Webanwendung sind Übungen. Eine Übung repräsentiert immer eine Aufgabensammlung, welche aus mindestens einer Aufgabe besteht. Neben der Aufgabenstellung an sich sollte jede Aufgabe auch Hinweise und unterstützende Informationen enthalten, die zu schnellen und einfachen Lösungen beitragen. Mehrere Übungen sind zu sogenannten Kursen zusammengefasst und stellen ein in sich geschlossenes Curriculum dar. Darüber hinaus besitzt jede Übung eine Sammlung an Metadaten, mit Inhalten wie zum Beispiel Titel, Autor, Erstellungsdatum, etc..

Erstellung von Übungen

Benutzer der zu entwickelnden Webanwendung können Übungen und damit auch Kurse erstellen. Hierfür muss der jeweilige Benutzer jedoch als Autor zu autorisieren sein. Für die Erstellung einer Übung sollte nicht mehr als eine View eingeplant werden, da dieser Prozess sehr einfach gehalten werden soll. Wenn ein Benutzer mindestens eine Übung, mit mindestens einer Aufgabe angelegt hat, muss er diese in einem Kurs zusammenfassen bzw. einem bestehenden Kurs zuordnen.

3 Theoretischer Entwurf der Webanwendung

Bearbeitung von Übungen

Selbstverständlich ist es einem Autor möglich, bereits vorhandene Übungen in einer geeigneten View komplett zu überarbeiten. Es werden stets die Daten des Datensatzes aktualisiert und kein neuer Datensatz erzeugt.

Verwaltung von Übungen in Kursen

Die erstellten Übungen müssen Kursen zugeordnet sein. Jeder Kurs ist ausschließlich eine Sammlung an Übungen, mit minimalistischen Metadaten. Innerhalb eines Kurses besitzt jede Übung eine, mehrere oder keine Folgeübung, sodass nach einer erfolgreich abgeschlossenen Übung zur nächsten Übung verwiesen werden kann. Wenn keine Folgeübung definiert wurde, stellt diese Übung das Kursende dar. Dieser Zusammenhang beschreibt die Abbildung des Curriculums.

Validierung von Übungen

Jede Übung muss nach dem Lösen der Aufgaben validiert werden. Für diese Validierung muss der vom Benutzer erstellte Quelltext von der Webanwendung interpretiert werden und auf syntaktische, sowie semantische Korrektheit überprüft werden. Unabhängig vom Erfolg muss das Ergebnis dem Client übergeben und angezeigt werden. Darüber hinaus müssen bei Erfolg bestimmte Prozesse die Übung, in Bezug auf den jeweiligen Benutzer, als gelöst markieren und dem Benutzer signalisieren, dass er im Kurs zur nächsten Übung fortschreiten kann.

Löschen von Übungen

In der Datenbank abgelegte Übungen können von autorisierten Benutzern zu jeder Zeit gelöscht werden. Da diese Operation alle zugehörigen Daten,

3 Theoretischer Entwurf der Webanwendung

wie zum Beispiel »Kursabhängigkeiten« endgültig und unwiderruflich aus der Datenbank entfernt, muss der Benutzer die Durchführung der Löschung bestätigen.

Authentifizierung von Benutzern

Das Benutzermanagement sollte extern über Open-ID erfolgen. Die dafür notwendigen Schnittstellen, stehen bei der Entwicklung mit dem Play Framework bereits zur Verfügung. Einzig die Benutzerinformation zur Autorisierung und zu Übungen sollten von der Webanwendung gehalten werden.

Autorisierung von Benutzern

Zur Verwaltung der Autorisierung und Erteilung von Zugriffsrechten auf Operationen implementiert die Webanwendung ein minimalistisches Rollensystem, welches auf der Authentifizierung von Benutzern basiert und zwischen folgenden Gruppen unterscheidet:

1. **Benutzer** sind Nutzer, die keine besonderen Rechte besitzen. Sie dürfen Kurse *ausschließlich* absolvieren.
2. **Autoren** sind Nutzer, welche die Rechte besitzen Übungen und Kurse zu erstellen bzw. zu verwalten.
3. **Administratoren** besitzen alle Zugriffsrechte.

Benutzern der Gruppe »Benutzer« ist es möglich, Kurse und deren Übungen durchzuführen, jedoch keine zu erstellen. Authentifizierten Benutzern der Gruppe »Autoren« ist es hingegen erlaubt, Übungen in der Datenbank abzulegen, zu bearbeiten und wieder zu löschen. Wobei das Bearbeiten und Löschen bei regulären Autoren auf die Menge der selbst erstellten Übungen beschränkt ist. Über alle Zugriffsrechte verfügen die Benutzer der Gruppe der »Administratoren«. Diesen ist es erlaubt Übungsdaten und Benutzerdaten zu verwalten.

3 Theoretischer Entwurf der Webanwendung

Die Administratoren sind für die Verwaltung der Webanwendung zuständig. Dies beinhaltet insbesondere die Pflege und die Qualitätssicherung der angelegten Übungen bzw. Kurse.

3.1.2 Nichtfunktionale Anforderungen

Zusätzlich zu den Umsetzungen der funktionalen Anforderungen, sollen für die Realisierung der Webanwendung folgende Rahmenbedingungen berücksichtigt werden:

Design, Layout und Usability

Die Webanwendung sollte sich stets an einem vorher festgelegtem Designmuster orientieren, sodass der Benutzer nach kurzer Zeit mit der Benutzeroberfläche vertraut wird. Das Layout, welches für die Haupt-View gedacht ist, sollte benutzerfreundlich sein. Das heißt die einzelnen Bereiche, wie zum Beispiel die Bereiche Aufgaben, Editor, Ausgabe/Konsole, Hilfe- und Informationstexte, klar voneinander zu trennen und zu präsentieren.

Bei der Konzeption der gesamten **GUI** müssen folgende Aspekte berücksichtigt werden:

1. Verwendung eines primitiven Rahmenlayouts für die Hauptnavigation.
2. Unterseiten, die relativ wenig Inhalt enthalten, sollten im Pop-up Fenster dargestellt werden, zum Beispiel: Impressum, Kontakt, etc..
3. Die Navigationstiefe sollte gering gehalten werden, um den Navigationsaufwand benutzerfreundlich gering zu halten.
4. Intelligent angelegte Tooltips, Hinweise und Hilfestellungen sollten an den Stellen vorhanden sein, wo klar zu erkennen ist, dass Probleme entstehen könnten.

3 Theoretischer Entwurf der Webanwendung

Skalierbarkeit des Datenmodells

Das Datenmodell der Übungen, Kurse, Aufgaben, etc. muss gewährleisten, dass eine nachträgliche Erweiterung ohne großen Aufwand, in Bezug auf Datenmodellierung, möglich ist. Es soll also sichergestellt sein, dass die Anwendung, zu jedem Zeitpunkt um neue Funktionalitäten erweiterbar ist.

Entwicklung in Java unter Verwendung des Play Frameworks

Wie zu Beginn erwähnt, soll die Anwendung als Webanwendung unter Verwendung der objektorientierten Programmiersprache Java (siehe Abschnitt 2.1) und des Java Play Frameworks (siehe Abschnitt 2.2) realisiert werden.

3.1.3 Tabellarische Zusammenfassung

In der folgenden Tabelle (siehe Seite 39) sind die eben erarbeiteten funktionalen und nicht funktionalen Anforderungen zusammengefasst und durch Identifikatoren benannt. Diese Identifikatoren werden sowohl teilweise im folgenden Abschnitt als auch letztlich vollständig in dem zusammenfassenden Komponenten-Diagramm (siehe Abbildung 3.2 auf Seite 44) wiederkehren.

3 Theoretischer Entwurf der Webanwendung

Identifikator	Kurzbeschreibung
	Übungen als zentrale Objekte
F1.1	Verknüpfung mit Aufgaben
F1.2	Verknüpfung zu Metadaten
F1.3	Verknüpfung zu einem Curriculum
	Erstellung von Übungen
F2.1	Erstellung von Aufgaben und deren Sortierung
F2.2	Erstellung von Metadaten
F2.3	Erstellung der korrekten Validierung bzw. der Lösung
	Bearbeitung von Übungen
F3.1	Bearbeitung der Metadaten
F3.2	Bearbeitung der Aufgaben
F3.3	Bearbeitung der Validierung bzw. der Lösung
F3.4	Validierung während der Bearbeitung möglich
	Verwaltung von Übungen in Kursen (Curriculum)
F4.1	Suchfunktion über bestimmte Metadaten
F4.2	Verwalten von Übungen in Kursen
F4.3	Bestimmung der Übungsposition im Curriculum
	Validierung von Übungen
F5.1	Validierung der Eingabe über Lösungsmuster
F5.2	Auswertung des Ergebnisses in der selben Benutzeroberfläche
F5.3	Speichern des Fortschritts
	Löschen von Übungen
F6.1	Löschen einer Übung und allen damit verbundenen Aufgaben und Curriculumsbeziehung
	Authentifizierung von Benutzern
F7.1	Authentitätsverwaltung von Benutzerkonten (Open-Id + Intern)
	Autorisierung von Benutzern
F8.1	Realisierung eines Rollensystems (Benutzer/ Autor/ Administrator)
	Verwaltung von Benutzerkonten
F9.1	Administration der Benutzerkonten
	Nichtfunktionale Anforderungen
NF1	Benutzerfreundliche Weboberflächen
NF2	Skalierbares dynamisches Datenmodell
NF3	Entwicklung in Java unter Verwendung des Play Frameworks

3.2 Identifikation der funktionalen Komponenten

Die Anforderungsanalyse unter Abschnitt 3.1 zeigt, dass die zu entwickelnde Webanwendung grundlegende Operationen zur Entwicklung von Curricula mit Aufgaben, Übungen und Kursen bereitstellen soll. Die Operationen sind wie folgt aufgelistet:

1. Die Erstellung von Aufgaben, Übungen und Kursen (eng. *create*).
2. Das Darstellen und Zusammenfassen von Aufgaben, Übungen und Kursen (eng. *read*).
3. Das Editieren von Aufgaben, Übungen und Kursen (eng. *update*).
4. Das Löschen von Aufgaben, Übungen und Kursen (eng. *delete*).

Diese vier Grundoperationen werden auch im Zusammenhang mit Datenbanken als **CRUD** bezeichnet.

Als weitere zentrale Anforderung gilt die Validierung einer Übung. Obwohl die Validierung von Übungen auf den ersten Blick keiner **CRUD** Operation zugeordnet werden kann, lässt sie sich mit den Punkten 1. und 3. verbinden. Die Validierung soll, wie in F3.4 gefordert während der Bearbeitung einer Übung ausführbar sein. Das heißt, dass ein Autor oder Administrator während der Bearbeitung einer Übung stets die Bearbeitung der Übung durch einen Lerner simulieren kann. Das hat den Vorteil, dass die Erstellung einer Übung bzw. deren einzelnen Aufgaben schnell geht und damit sehr benutzerfreundlich bleibt. Die Anwendung unterscheidet bei der Validierung nicht, ob der zu betrachtende Code von einer Erstellung oder von einer Bearbeitung durch einen Lerner kommt. In beiden Fällen wird der Code lediglich validiert und das Ergebnis zurückgesendet.

3 Theoretischer Entwurf der Webanwendung

Die Validierung einer Übung, ob für die Erstellung, Bearbeitung oder Bearbeitung durch den Lerner, lässt sich also zweckmäßig in eine einzige funktionale Komponente realisieren. Für diese Validierung muss demnach auch nur eine zentrale Validierungs-Klasse zur Verfügung stehen.

In Bezug auf die Benutzeroberfläche ist festzuhalten, dass für die Erstellung und Bearbeitung von Übungen durch einen Autor oder Administrator auch nur eine zentrale Weboberfläche notwendig ist, da sich Erstellung und Bearbeitung im Hinblick auf die Benutzerschnittstelle nicht unterscheiden. Bei einer Bearbeitung kommt lediglich die Funktion des Löschens hinzu (siehe Punkt 4.).

Die eben beschriebene Validierungs-Komponente ist zudem exklusiv für die Validierung von Code zuständig und stets unabhängig von anderen funktionalen Komponenten der Webanwendung. Diese strikt modulare Herangehensweise fördert die Skalierbarkeit, Wartbarkeit und Übersichtlichkeit der Software. Im Weiteren wird diese Komponente ausschließlich Validierungskomponente genannt.

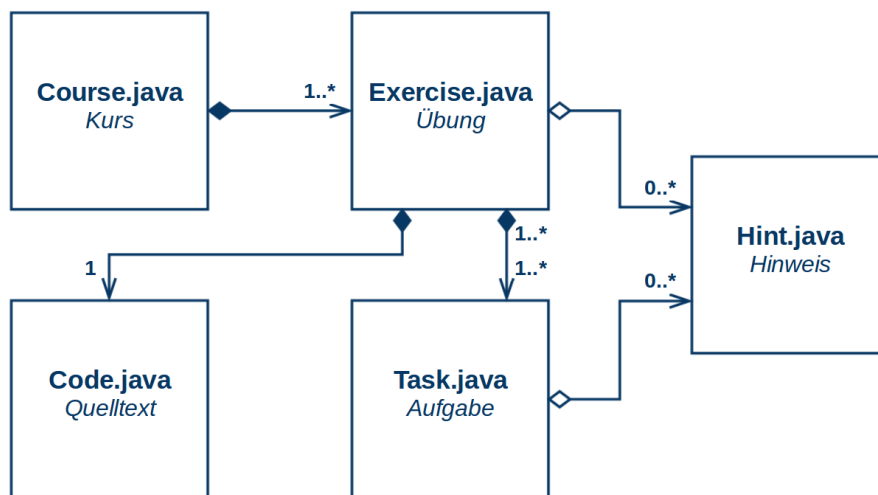


Abbildung 3.1: Klassendiagramm - UML

3 Theoretischer Entwurf der Webanwendung

Die Darstellung und Aufbereitung der Übungen (siehe Punkt 2. der [CRUD-Operationen](#)) wird in einer weiteren funktionalen Komponente zusammengefasst. Übungen sind, wie bereits erwähnt, Kursen zugeordnet und sind auch über diese erreichbar. So ergibt sich ein Mapping, was wie folgt aussieht:

Einem Benutzer stehen alle Kurse in einer Liste zusammengefasst zur Verfügung, wobei die Listensortierung stets adaptierbar bleibt. Nach der Auswahl eines Kurses bekommt der Benutzer eine neue Liste angezeigt, welche alle unter dem Kurs liegenden Übungen auflistet. Ob eine Übung in dieser Liste als aktiv bzw. inaktiv (noch gesperrt) markiert ist, hängt vom Fortschrittsstatus des Benutzers ab. Dieser Fortschrittsstatus wird in einer Datenbanktabelle festgehalten, welche die Beziehungen zwischen Benutzern und Curricula-Objekten festhält. Hierbei wird deutlich das Ziel verfolgt, das Curriculum in der Weboberfläche in Form von Listen- und Navigationselementen darzustellen. Dem Benutzer dürfen keine Probleme bei der Erkennung des eigenen Fortschritts in dem jeweiligen Curriculum aufkommen. Diese funktionale Komponente, der Darstellung des Curriculums als Kurs-Navigation bzw. Einstiegsübersicht, wird im folgenden als Kursübersicht bezeichnet.

Ein Benutzer der Gruppe »Benutzer« (F8.1) muss über Kurse zu den jeweiligen Übungen navigieren. Alternativ kann jeder Benutzer auch eine separate Suchfunktion (F4.1) nutzen, welche jedoch ausschließlich eigene Übungen, bereits erfolgreich gelöste Übungen und weitere freie Übungen liefert. Ob eine Übung für einen bestimmten Benutzer sichtbar ist, hängt also vom jeweiligen Curriculum und dem Status — welche Übungen hat der Benutzer in diesem Kurs/Curriculum bereits absolviert — des Benutzers ab.

Bei der bereits angesprochenen Bearbeitung von Übungen kommt eine weitere funktionale Komponente zum Einsatz, die Editor-Komponente. Die Weboberfläche des Editors soll als interaktiver Editor verstanden werden. Diese Komponente besteht aus vier Modulen. Dem Editor, der Ausgabe, die Übung — mit ihren einzelnen Aufgaben — und der Hauptnavigation, welche über allen Unternavigationen steht und unabhängig von Übungen und Kursen ist. Die Aufgaben sind einer Liste mit Anmerkungen zu entnehmen, welche unter der Überschrift

3 Theoretischer Entwurf der Webanwendung

der Übung zu finden ist. Alle Aufgaben sind innerhalb des Editor in Form von Programmierungen zu lösen. Die Bestätigung des Abschlusses aller Aufgaben wird mittels »Shortcuts« oder einem »Kompilieren & Ausführen«-Button durchgeführt. Die von der Validierungs-Komponente kommenden Antworten werden in dem Modul »Ausgabe«, und auch als Popup angezeigt. Das Modul »Ausgabe«, soll sich an einem klassischen Terminal orientieren und gibt deshalb die Antworten des Compilers zurück. In dem erwähnten Popup sollen bei positiver Validierung weiterführende Texte und bei negativer Validierung eventuell Hilfetexte angezeigt werden.

Neben den drei zentralen funktionalen Komponenten, der Validierungs-, der Kursübersicht- und Editor-Komponente wird die Anwendung noch um drei weitere Komponenten ergänzt:

- Eine Authentifizierungs-Komponente, welche die Verwaltung und Authentifizierung von Benutzern implementiert.
- Eine Administrations-Komponente, die alle Funktionalitäten zur Administration der Webanwendung implementiert.
- Eine Autorisierungs-Komponente, die ein Rollensystem aufsetzt und alle Funktionalität zur Autorisierung implementiert.

Die eben aufgeführten zusätzlichen Komponenten werden auch explizit im vorherigen Abschnitt unter den funktionalen Anforderungen F7.1, F8.1 und F9.1 gefordert. Die Authentifizierungs-Komponente umfasst alle Funktionen für die Registrierung, Bearbeitung und Löschung von Benutzern und die Anmeldung am System. Diese Anmeldung innerhalb der Anwendung kann über den eigenen Benutzerbestand oder über externe Bestände, via Open-ID Schnittstelle erfolgen. Der Grund für die Trennung von Administration und Autorisierung ist, dass einem autorisierten Benutzer nicht zugleich zwingend alle Funktionalitäten der Administration zur Verfügung stehen. Zudem soll die Administration ihren Schwerpunkt in der Inhaltsverwaltung und nicht in der Benutzerverwaltung haben.

3.3 Aufbau und Funktionsweise der Webanwendung

Die Abbildung 3.2 soll einen Überblick über den modularen Aufbau der zu entwickelnden Webanwendung geben. Sie zeigt zusätzlich zur Interaktion der Module zueinander die jeweiligen funktionalen Anforderungen (siehe Abschnitt 3.1.3) jeder Komponente.

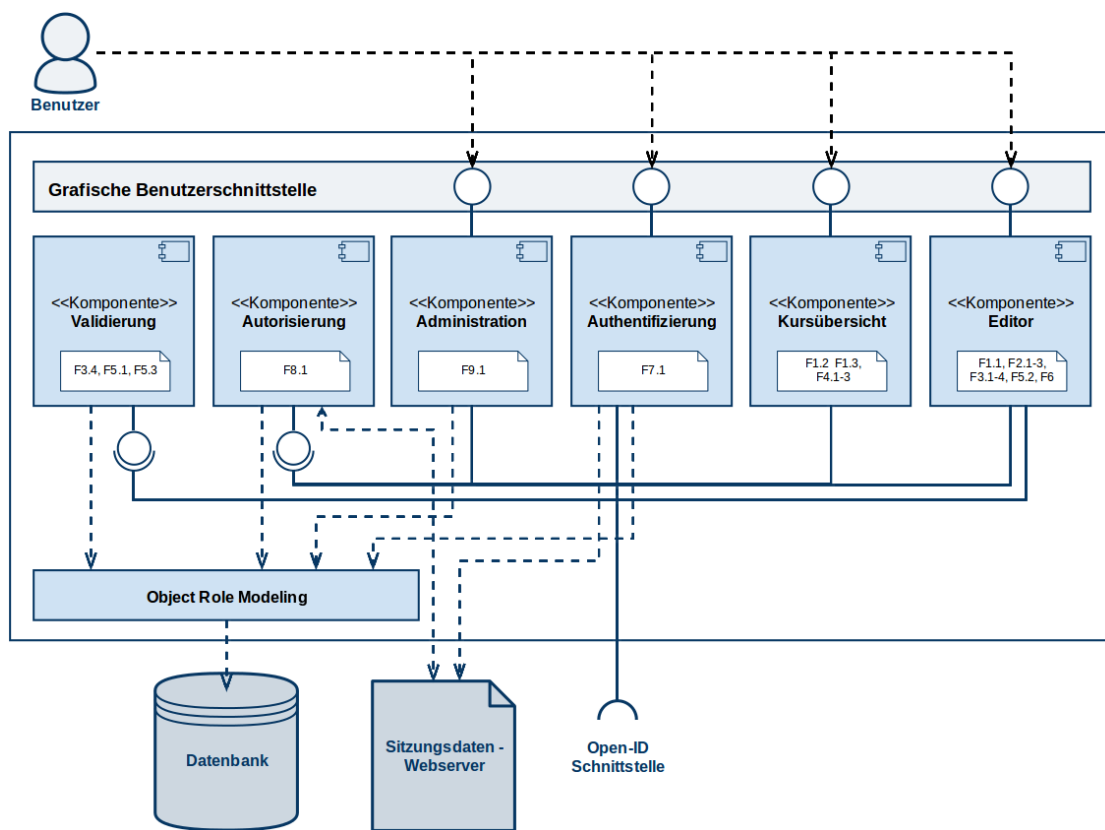


Abbildung 3.2: Komponenten Diagramm

Die grafische Benutzeroberfläche (GUI) der Webanwendung wird durch die Front-Ends (Views) bestimmter Komponenten gebildet. Diese Benutzeroberflächen sind über einen Webbrowser anzusprechen. Die Komponenten Kurs-

3 Theoretischer Entwurf der Webanwendung

übersicht, Editor, Authentifizierung und Administration liefern eine grafische Benutzerschnittstelle, wobei die Komponenten der Autorisierung und der Validierung dem Benutzer eine ausschließlich funktionale Schnittstelle bereitstellen. Die Funktionen dieser Schnittstellen sind ausschließlich über andere Komponenten, welche eine Benutzeroberfläche bereitstellen, nutzbar. Zum Beispiel nutzt die Administrationskomponente Funktionen sowie der Authentifizierungs- als auch der Autorisierungskomponente, um dem Benutzer abhängig von Zugriffsrechten bestimmte Funktionalitäten einzuräumen. Diese Benutzer- und Rechedaten für eine Sitzung werden von den beiden Komponenten (Authentifizierung und Autorisierung) in den Sitzungsdaten des Webserver verwaltet. Für eine persistente Speicherung der Daten dient die Datenbank.

Aktionen in den grafischen Benutzeroberflächen ([HTTP](#) Request/Post) werden immer von dem Controller der jeweiligen Komponente ausgewertet. Das heißt im Umkehrschluss, dass ausschließlich die Komponenten, die eine grafische Benutzerschnittstelle bereitstellen, eine View und einen Controller implementiert haben, welcher über das Routing (»Play Framework - Routing« wird im Kapitel 4 näher erläutert) ansprechbar ist. Wenn ein Controller temporäre Daten bzw. Objekte in die Datenbank speichern will, geschieht das stets über die jeweilige Klasse des Objekts, welche von Ebean-Models erbt. Dies hat den großen Vorteil, dass eine Datenkonsistenz gesichert ist. Damit sind besonders Datentypen und Datenbeziehungen gemeint, welche somit typensicher vorliegen sollten. Nach Bearbeitung der Anfrage übergibt der Controller alle anzuzeigenden Ergebnisse bzw. Daten der jeweiligen View der Komponente, welche dann wiederum dem Benutzer als Antwort ([HTTP](#) Response) an den Webbrowser gesendet werden.

Im Folgenden werden die einzelnen grafischen Benutzeroberflächen der Komponenten mit Hilfe von Mockups, welche die wichtigsten Elemente zeigen und näheren Erläuterungen der Funktionen beinhalten, dargestellt. Die Mockups sollen *ausschließlich* als grobe Orientierung für das spätere Layout-Design dienen, sodass eventuell Anordnungen, Labels, Größen, Farben, Grafiken, etc. von dem zu entwickelndem Prototyp abweichen können.

3 Theoretischer Entwurf der Webanwendung

3.3.1 Editor Komponente

Abbildung 3.3 zeigt einen Entwurf von der Benutzeroberfläche der Editor-Komponente (im folgenden nur noch als Editor bezeichnet) im Webbrowser.

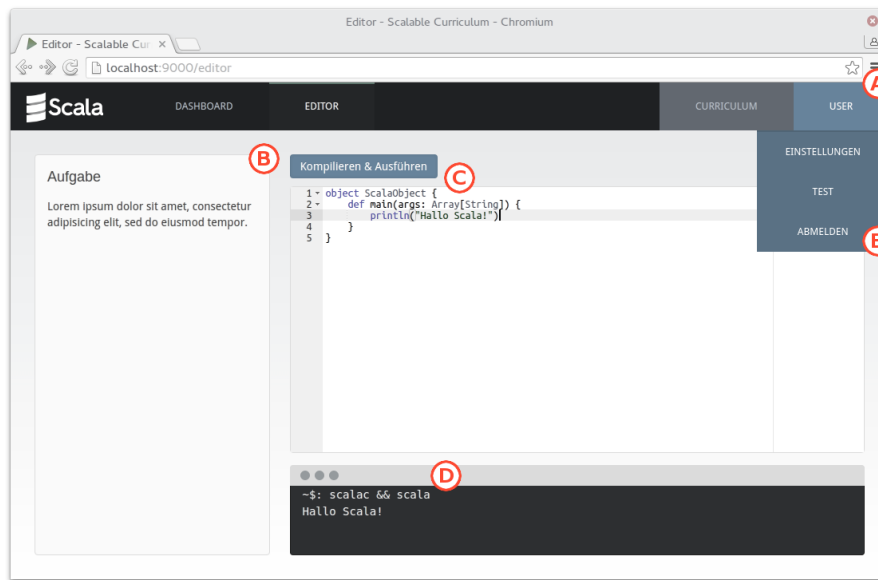


Abbildung 3.3: Editor (Entwurf)

Den Einstieg in die verschiedenen Komponenten ermöglicht die Navigationsleiste (A), welche das wesentlichste Element des Rahmenlayouts darstellt. Sie sollte unter jeder Komponenten-View das selbe Layout und die gleichen Farben aufweisen, sodass nur die Inhalte (die Navigationspunkte) variieren. In der rechten Ecke der Navigationsleiste befindet sich das Benutzer-Dropdown Menü, welches mit dessen Inhalt abhängig von den Sitzungsdaten ist. Sollte kein Benutzer angemeldet sein, verweist dieses Menü zur »Anmeldung« bzw. »Registrierung«. Es kann also festgehalten werden, dass alles unterhalb der Navigationsleiste ein komponentenabhängiger Kontext ist und demnach als dynamischer Inhalt gilt.

Eine Übung mit Aufgaben

In der linken Spalte Übung (B) wird dem Benutzer die aktuelle Übung mit den dazugehörigen Aufgaben präsentiert. Unter jeder Übung befindet sich ein einleitender Text, welcher durch Bilder unterstützt werden kann. Dieser Text soll den Benutzer in die Thematik, welches die Übung vermitteln möchte, einführen. Darunter folgen die einzelnen Aufgaben, welche die Übung bilden. Eine Aufgabe kann verschiedene Hinweise besitzen, die dem Benutzer als Hilfestellung dienen sollen. Diese Hinweise können sowie aus Texten, als auch aus Bildern bestehen. Alle Hinweise werden durch Autoren verfasst, bearbeitet und gelöscht, jedoch ist dies nur über die Administrationskomponente möglich (siehe Abschnitt 3.3.4).

Zusätzlich befinden sich in der linken Spalte weiterführende Verweise, so zum Beispiel »Problem melden« oder »über den Autor der Übung« . Um stets eine Übersichtlichkeit und eine Selektion einzelner Aufgaben zu gewährleisten, muss die gesamte Spalte als Akkordionelement implementiert werden. Dies hat auch den positiven Nebeneffekt, dass die Informationsdichte erhöht und die zu scrollenden Abschnitte verringert werden.

Editor

Nachdem sich der Benutzer in das Thema eingelesen und die genauen Aufgabenstellungen verstanden hat, muss er alle Aufgaben lösen, um die Übung erfolgreich zu absolvieren. Um eine Aufgabe zu lösen, muss vom Benutzer ein Programmcode, der von einem einfachen »Codeschnippel« bis zu einem komplexen Algorithmus alles darstellen kann, implementiert werden. Dieser Programmcode soll mit Hilfe des Editors (C) geschrieben werden. Der Editor basiert auf Ace¹, einem freien, in Javascript geschriebenen Texteditor, welcher für die Anwendung im Web prädestiniert ist. Ace zeichnet sich durch Eigenschaften wie Syntaxheighlighting, Zeilennummerierung, Automatische Codeeintrückung,

¹ siehe ace.c9.io [1, Offizielle Webseite]

3 Theoretischer Entwurf der Webanwendung

Tastenkürzel, Suchen und Ersetzen-Funktion und Code-Einklapp-Funktion aus.

Jede Übung besitzt genau einen »default«-Code, welcher dem Benutzer einen Lösungsansatz andeuten soll. Wenn ein Benutzer zu einer Übung bereits an einer Lösung gearbeitet hat und diesen Code via »Kompilieren & Ausführen« abgeschickt hat, wird dieser dem Benutzer in Zukunft immer anstelle des »default«-Codes angezeigt. Selbstverständlich hat der Benutzer zu jedem Zeitpunkt die Möglichkeit, den Code bzw. die Übung zurückzusetzen und seinen bis dato geschriebenen Code zu löschen und mit dem »default«-Codes erneut zu beginnen.

Ausgabe

Mit dem Button »Kompilieren & Ausführen« sendet der Benutzer die Lösung einer Übung — den Quelltext im Ace-Editor — an den Server und somit über die Editorkomponente zur Validierungskomponente. Die Validierungskomponente kompiliert den Code auf dem Server mittels des Scala Compilers und führt diesen, falls der Bytecode erfolgreich erzeugt wurde, aus. Die Ausgabe der JVM wird in einen Buffer geschrieben und direkt über den Editorkomponenten-Controller in der View neu gerendert und an den Webbrowser des Benutzers zurückgeschickt. Der Benutzer kann demnach die Systemausgabe, welche sein Code erzeugt, in der Ausgabe (D) nachvollziehen. Falls das Kompilieren des Codes fehlschlägt, zeigt die Ausgabe sämtliche Compiler-Errors an, sodass ein Debugging seitens des Benutzers immer möglich ist. Ein Benutzer kann sich die Historie der Ausgabe in einem Popup-Fenster anzeigen lassen. Zu beachten ist jedoch, dass es sich bei der Historie um alle Ausgaben einer bestimmten gerade aktuellen Übung und einer aktuellen Sitzung handelt. Demnach wird ausschließlich der Code und nicht die Ausgaben in der Datenbank konstant abgespeichert.

3 Theoretischer Entwurf der Webanwendung

Popup

Bei einer erfolgreichen Validierung des Codes, das heißt die Ausgabe und der geschriebene Code erfüllen die Anforderungen einer Übung, wird dem Benutzer ein Popup-Fenster angezeigt, welches den erfolgreichen Abschluss der Übung bestätigt. Nach Abschluss einer Übung kann man zur nächsten Aufgabe navigieren oder das Popupfenster schließen und weiter an der gelösten Übung programmieren. Zu beachten ist, dass nach einem Abschluss einer Übung der Code noch editiert und abgeschickt, also kompiliert und ausgeführt werden kann. Jedoch wird dieser dann nicht mehr auf Erfüllung der Aufgabenstellungen validiert. Somit ist es dem Benutzer freigestellt seinen Code im Nachgang zu bearbeiten.

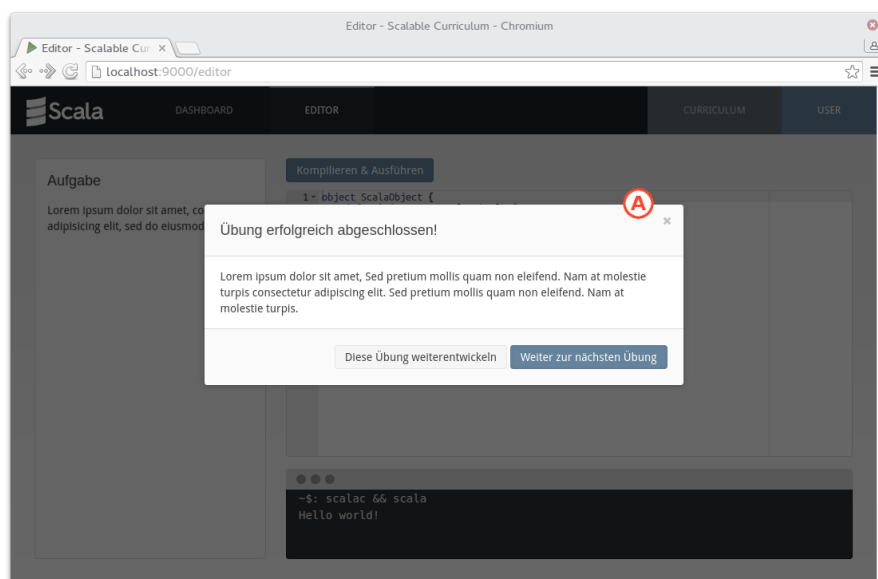


Abbildung 3.4: Popup (Entwurf)

Das Popup-Fenster sollte sich genau wie das Ausgabehistorie-Popup-Fenster einem einheitlichen Popup-Layout bedienen, um einen konstanten Stil aufzuzeigen.

Benutzermenü

Das Benutzermenü (E), welches immer über die Navigationsleiste erreichbar ist, vorausgesetzt ein Benutzer ist zu einer Sitzung angemeldet, verweist auf die wichtigsten Seiten und Funktionen bezüglich des Benutzermanagments. Zum Beispiel »Mein Profil« , »Einstellungen« und »Abmelden« , aber auch Allgemeine Hilfe oder Support sind hier zu platzieren.

3.3.2 Kursübersicht

Die Kursübersicht soll dem Benutzer einen Einstieg in die Übungsserien ermöglichen. Hierfür listet diese zunächst alle Kurse auf. Unter jedem Kurs kann man die jeweiligen Übungen in der Reihenfolge, wie sie von dem Curriculum bestimmt wird, sehen. Zu jedem Kurs werden neben den Grunddaten wie Name, Autor und Schwierigkeitsgrad auch statistische Daten, wie zum Beispiel Aufrufe der einzelnen Übungen oder wie oft ein Kurs erfolgreich abgeschlossen wurde, dargestellt.

Aus Sicht des Benutzers

Für den Benutzer ist die Kursübersicht zugleich eine Leistungs- bzw. Fortschrittsanzeige. Diese weist auch zu jedem Kurs den jeweiligen Fortschrittsstand in Prozent aus. Die Übungen, welche der Benutzer bereits gelöst hat, sind mit einem Symbol versehen, welches diesen Status repräsentiert. Übungen, die noch nicht gelöst wurden, jedoch möglich sind zu lösen, sind farbig und durch kein Symbol gekennzeichnet. Noch gesperrte Übungen sind ausgegraut bzw. sind ersichtlich als noch gesperrt zu erkennen.

Aus Sicht des Administrators/Autors

Wenn der Benutzer als Autor oder Administrator autorisiert ist, hat er die Möglichkeit, in der Kursübersicht neue Kurse anzulegen, bestehende Kurse zu bearbeiten oder Kurse zu löschen. Mit Administrationsrechten sind die Zugriffe auf alle Kurse ohne Einschränkung möglich. Dem Autor ist es nur gestattet eigene Kurse² zu bearbeiten. Unter Voraussetzung dieser Rechte ist es dem Benutzer auch erlaubt, das Curriculum eines Kurses aus der Kursübersicht heraus zu bearbeiten. Hierfür stehen ihm zwei Funktionen zur Verfügung. Die Übungen können entweder über Pfeil-Buttons Ihre Position innerhalb eines Kurses ändern oder via Drag and Drop gleich über mehrere Positionen verschoben werden. Zu beachten ist, dass eine Übung stets unter dem selben Kurs geordnet ist und nicht in einen anderen Kurs verschoben werden kann.

3.3.3 Authentifizierung

Registrierung

Die Registrierung eines Benutzers ist notwendig, um den jeweiligen Benutzer persistent zu speichern und ihm stets seine persönlichen Daten³ bereitzustellen. Dies ermöglicht dem Benutzer ein Anwenden, indem die Daten nicht auf eine bestimmte Sitzung begrenzt sind, sondern ein Anwenden über mehrere Tage, Wochen oder Monate, bei dem alle wichtigen Daten in einer Datenbank gehalten werden.

Für die Registrierung steht eine extra View zur Verfügung, welche über einen Button in der Anmelden-View (siehe Abbildung 3.6 auf Seite 53) erreichbar ist.

² Das betrifft Kurse bei denen der Benutzer als Autor eingetragen ist.

³ Als persönliche Daten gelten Übungsfortschritt, Profil und Berechtigungen.

3 Theoretischer Entwurf der Webanwendung

The screenshot shows a web browser window with the title 'Registrieren - Scalable Curriculum - Chromium'. The address bar shows 'localhost:9000/register'. The page has a dark header with the 'Scala' logo on the left and 'CURRICULUM' and 'ANMELDEN' links on the right. The main content area is light grey and contains a white registration form. The form has the title 'Registrieren' and three input fields: 'Benutzername', 'E-Mail-Adresse', and 'Passwort'. Below these fields are two buttons: a green 'Registrieren' button and a dark grey 'Zurück zur Anmeldung' button. At the bottom of the form, there is a small blue link that says 'Benötigen Sie Hilfe?'.

Abbildung 3.5: Registration (Entwurf)

Diese View beinhaltet neben der Hauptnavigation eine Form, mit drei Feldern: Benutzername, E-Mail-Adresse und Passwort. Mehr Daten sind zur Registrierung nicht notwendig. Vorausgesetzt, dass die E-Mail-Adresse und der Benutzername noch nicht im System registriert sind, wird nach dem Absenden dieser Form, über den Button »Registrieren«, der Benutzer am System registriert und zugleich automatisch angemeldet.

Anmeldung

In Folge der Registrierung kann der Benutzer sich immer über die View der Anmeldung mit E-Mail-Adresse und Passwort authentifizieren. Falls während einer Sitzung noch keine Authentifizierung stattgefunden hat, wird in der Navigationsleiste ein Verweis zur Anmeldung⁴ aufgeführt.

⁴ Nach erfolgreicher Authentifizierung ist das Benutzermenü verfügbar.

3 Theoretischer Entwurf der Webanwendung

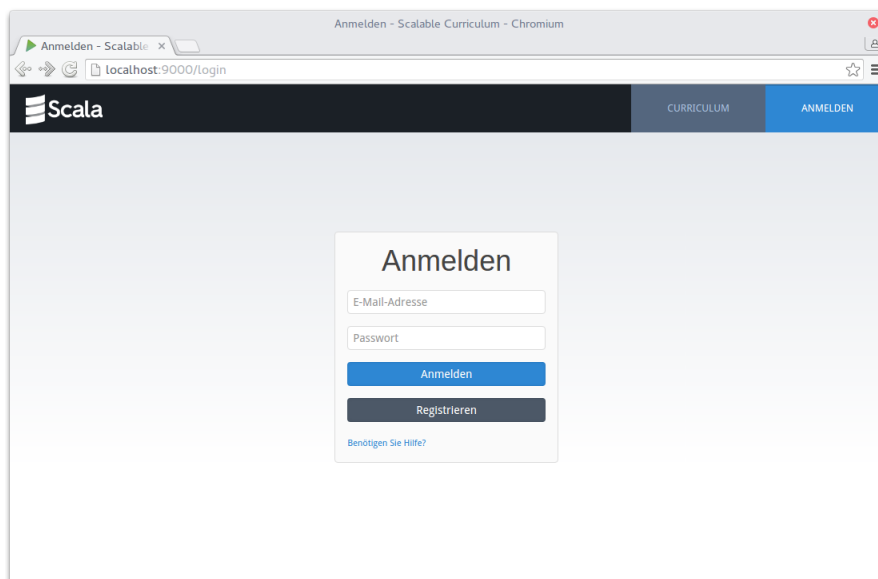


Abbildung 3.6: Anmeldung (Entwurf)

Über den Link »Benötigen Sie Hilfe?« sollen dem Benutzer einige häufig gestellten Fragen beantwortet werden. Zudem soll dem Benutzer unter Angabe seiner E-Mail-Adresse die Möglichkeit gegeben werden sein Passwort, falls dieses nicht mehr bekannt ist, zurückzusetzen. In einer Bestätigungsmail bekommt er ein neues temporäres Passwort zugeschickt.

3.3.4 Administration

Alle administrativen Funktionen stellt die Administrations-Komponente bereit. Diese besitzt eine eigene View (im folgendem als Adminbereich ausgeschrieben) und ist ausschließlich für Benutzer erreichbar, welche als Administrator autorisiert sind. Innerhalb des Adminbereichs sind alle registrierten Benutzer mit deren Daten tabellarisch aufgelistet. Aus dieser Tabelle heraus kann einem Benutzer eine bestimmte Systemrolle⁵ zugewiesen werden. Die Tabelle soll als

⁵ Systemrollen bilden das Rechtssystem innerhalb der Webanwendung.

3 Theoretischer Entwurf der Webanwendung

zentrales Element der Administration dienen, sodass alle administrativen Funktionen aus ihr heraus nutzbar sind. Des Weiteren soll sie durch alle Attribute sortierbar und über ein extra Suchfeld durchsuchbar sein.















User Management						
Filter <input type="text"/>						Create New User
Full name	Email	User name	Access level	Status	Last login	
Vilmos Fiatal	vilmos.fiatal@example.com	vilmos	Administrator	Pending	2016-02-25 09:32:50	  
Bob Bonsai	bob.bonsai@example.com	Jeff	Administrator	Pending		  
Diana Prince	will.young@emarsys.com	DPrince	Account Administrator	Active	2016-02-29 08:43:51	
Hagai Hartman	hagai.hartman@emarsys.com	hagai2	Administrator	Active		   
Pooh Bear	pbear@100acrewood.com	winniep	Administrator	Deactivated		  

Abbildung 3.7: Administration (Entwurf)

Die eben erarbeiteten Anforderungen sollen als Grundlage für eine prototypische Implementierung, der Webanwendung genügen.

4 Prototypische Implementierung der Webanwendung

Dieser Abschnitt beschreibt auszugsweise die prototypische Implementierung der Webanwendung »Scalable Curriculum«, unter Verwendung des in Kapitel 3 erarbeiteten Entwurfs. Zunächst werden die verwendeten Werkzeuge und Systeme vorgestellt. Unter dem Abschnitt 4.2 wird die Abbildung des konzeptinellen Aufbaus der Webanwendung (siehe hierzu Abbildung 3.2) in dem MVC-Architekturmuster im Play Framework beschrieben. Hierfür werden für jede Ebene Listings mit typischen Codepassagen näher kommentiert und deren Interaktion untereinander erklärt, sodass eine beispielhafte Benutzeraktion von der clientseitigen Anfrage, über deren Bearbeitung durch die Webanwendung, bis hin zurück zur serverseitigen Antwort nachvollzogen werden kann. Abschließend werden in diesem Kapitel die grundlegenden Klassen, die im Abschnitt 3.1.1 definierten funktionalen Anforderungen implementieren, sowohl in ihrem Aufbau, als auch in ihrer Funktionsweise beschrieben. Ein Schwerpunkt stellt hierbei die Klasse »Runsc« dar. Sie gilt als fundamentale Klasse, für die Implementierung der »Editorkomponente« (siehe Abschnitt 3.3.1).

4.1 Werkzeuge und Entwicklungsumgebungen

4.1.1 Server

Die Webanwendung wird, seit Beginn der Entwicklung, mit Git (siehe Abschnitt 2.4) versioniert, wobei stets drei Repositories auf drei verschiedenen Systemen unter unterschiedlichen Aspekten gehalten werden. Ein Repository befindet sich auf einem System, welches für Entwicklungsarbeiten genutzt wird; Ein weiteres Repository befindet sich auf einem Notebook, dass ausschließlich für Testzwecke gedacht ist und eine große Menge an vielen verschiedenen Webbrowsern installiert hat. Das Repository, das stets die stabilste und sauberste Implementierung der Webanwendung hält, wird unter Verwendung von Gogs¹ auf einem Virtuellen Server mit Linux CentOS, welcher von TwooIT gehostet wird und seinen Standort in Frankfurt hat, gepflegt. »Gogs (Go Git Service) ist ein einfacher selbst-gehosteter Git Service«², der mit einem bekannten Dienst, wie zum Beispiel GitHub³ oder Sourceforge⁴ zu vergleichen ist. Der beschriebene Dienst ist unter der URL »<http://git.kutzmann.eu>« erreichbar. Weiterführende Informationen und eine Anleitung zur Initialisierung des Projekts »Scalable Curriculum« sind im Anhang A zu finden.

Für die Speicherung der Arbeitsdaten der Webanwendung wurde nicht die speicherbasierte Datenbanklösung verwendet, die das Play Framework von Haus aus mitliefert, sondern ein separater MySQL-Server, welcher auf dem eben erwähnten Virtuellen Server läuft. Ein Grund für die Wahl eines von der Webanwendung unabhängigen Datenbank-Servers ist, dass alle Daten zum einen über den gesamten Entwicklungsprozess und zum anderen über alle einzelnen Entwicklungsumgebungen verfügbar sind.

¹ siehe gogs.io [23, Offizielle Webseite]

² siehe gogs.io [22, Dokumentation]

³ siehe github.com [20, Offizielle Webseite]

⁴ siehe sourceforge.net [46, Offizielle Webseite]

4.1.2 Entwicklungsumgebung

Dank Typesafes Activator⁵, der mit dem Play Framework zur Verfügung steht, ist die Entwicklung einer Webanwendung in Java mit Play auch komplett ohne eine IDE möglich und zugleich komfortabel. Dabei kann man den Activator entweder als Webanwendung starten und diesen wie eine Online-Entwicklungsumgebung nutzen oder als Kommandozeilenprogramm einzelne Befehle gezielt absetzen. Im Betrieb als Online-Entwicklungsumgebung stellt die Activator-Webanwendung Funktionen wie zum Beispiel »Projekt kompilieren« oder »Ausführen der Anwendung« *grafisch* bereit. Während der »Ausführung der Anwendung« präsentiert der Activator aktuelle Informationen über die laufende Anwendung. Für eine neue Projektinitialisierung bietet der Activator eine Vielzahl an Templates⁶ an. Bei der Implementierung des Projekts »Scalable Curriculum« wird der Activator jedoch ausschließlich als Kommandozeilenprogramm genutzt, da ein nativer Editor, welcher vom zugrunde liegendem System bereitgestellt wird, verwendet werden soll. Mit dem Befehl »activator run« wird die Webanwendung kompiliert und gestartet. Der integrierte Webserver Netty lauscht dann auf dem Standardport »9000«⁷. Wie bereits im Abschnitt 2.2.1 beschrieben, liegt ein großer Vorteil bei der Webentwicklung mit dem Play Framework in der Hot-Code-Reload-Funktion, die es erlaubt im Projekt zu programmieren und parallel die Anwendung auszuführen bzw. Auswirkungen der Änderungen nach erneutem Aufrufen der Webseiten sofort zu erkennen. Dies verdeutlicht die Unabhängigkeit zwischen Entwicklung und Ausführung der Webanwendung, was den Einsatz einer sehr *abgespeckten* IDE rechtfertigt.

Was die Programmierung an sich betrifft, konnten während der Entwicklungsphase sehr gute und effiziente Ergebnisse mit dem Editor Vim (auf Konsole-

⁵ siehe lightbend.com [27, Packages]

⁶ 458 Templates stehen für die Entwicklung mit dem Middleware-Framework Akka, dem Web-Framework Play und von reinen Scala-Programmen derzeit zur Verfügung (siehe lightbend.com [26, Activator Templates]).

⁷ Der Port »9000« kann in der Konfigurationsdatei »conf/application.conf« geändert werden.

4 Prototypische Implementierung der Webanwendung

nebene) und dem Editor Atom gemacht werden. Der Atom-Editor überzeugt durch optionale Pakete, wie zum Beispiel automatische Codevervollständigung für Java, Scala und dem UIKit-Framework⁸.

⁸ Atom-Packages: autocomplete-java, language-scala, atom-uikit (siehe atom.io [3, Packages])

4.2 Architektur mit Play

Wie in Abschnitt 2.2.1 erläutert, liegt dem Aufbau einer Webanwendung mit Play ein MVC-Architekturmuster zu Grunde. Die Strukturierung des MVC-Musters ist in dem, von dem Programm Activator initialisierten und in der weiteren Entwicklung ausgebauten Template deutlich zu erkennen. Ein Verzeichnisbaum eines beispielhaften Initialtemplates befindet sich im Anhang C. Im Anhang B wird der Verzeichnisbaum des Projekts »Scalable Curriculum« ausführlich beschrieben.

4.2.1 Models

Wie bereits unter dem Abschnitt 2.2.1 und dem Abschnitt 3.3 beschrieben, kommt mit der Verwendung des Play Frameworks das ORM-Framework Ebean zum Einsatz. Ebean gilt als besondere Alternative zu JPA-Implementierungen, wie zum Beispiel Hibernate. Ebean benutzt keine Session-Objekte (oder EntityManager), keine Attached oder Detached Beans, kein »merge()«, »persist()«, »flush()« oder »clear()«, sondern nur einfache »save()«- und »delete()«-Methoden. Zudem muss *ausschließlich* die Datenbank bereit gestellt werden. Sämtliche Tabellen werden, für eine korrekte Abbildung bzw. Speicherung der Objektdaten automatisch von Ebean erzeugt. Die Benutzerfreundlichkeit wird, durch die Verwendung von JPAs Assoziations-Annotationen gestützt. Sie ermöglichen mit einfachen Annotationen wie zum Beispiel »@OneToOne«, »@OneToMany«, »@ManyToOne« oder »@ManyToMany« alle notwendigen Assoziationstabellen automatisch zu erzeugen und zu pflegen. Dies ist vor allem bei der Entwicklung von »Scalable Curriculum«, wo fast alle Objekte miteinander in einer bestimmten Beziehung stehen, sehr effizient. Um ein besseres Verständnis des eben Erwähnten zu schaffen sind unter den Listings 4.1 und 4.2 zwei beispielhafte Quelltextauszüge zweier Modellklassen, welche sowohl eine direkte Assoziation zeigen, als auch die oben beschriebenen Methoden nutzen:

4 Prototypische Implementierung der Webanwendung

```
1 [...]
2 @Entity
3 @Table(name = "exercises")
4 public class Exercise extends Model {
5 [...]
6     @ManyToMany
7     public List<Task> tasks;
8     @ManyToOne
9     public Code code;
10 [...]
11     public void del() {
12         //Loesche alle zugewiesenen Aufgaben
13         for (Task task : this.tasks) {
14             task.exercises.remove(this);
15             task.save();
16             task.del();
17         }
18         //Loesche die Aufgabe selbst
19         this.delete();
20     }
21 }
```

Listing 4.1: Auszug des Model-Objekts Exercise (Übung)

Zunächst muss man ein Modelobjekt mit »@Entity«, als Datenbankobjekte annotieren (siehe hierfür jeweils die zweite Zeile, der beiden Listings 4.1 und 4.2). Da unterhalb einer Übung, wie der Abbildung 3.1 (siehe Seite 41) zu entnehmen ist, mehrere Aufgaben hängen können und eine Aufgabe in verschiedenen Übungen hängen kann, müssen diese Beziehungen abgebildet — im Englischen als Mapping bezeichnet — werden. Diese Assoziationen sind jeweils in den Zeilen 6 bis 9 definiert.

```
1 [...]
2 @Entity
3 @Table(name = "tasks")
4 public class Task extends Model {
5 [...]
6     @ManyToMany
7     public List<Hint> hints;
8     @ManyToMany(mappedBy = "tasks")
9     public List<Exercise> exercises;
10 [...]
```

4 Prototypische Implementierung der Webanwendung

```
11 public void del() {
12     //Loesche die Beziehung zu jeder Uebung
13     for (Exercise exercise : this.exercises) {
14         exercise.tasks.remove(this);
15         exercise.save();
16     }
17     //Loesche alle zugewiesenen Hinweise
18     for (Hint hint : this.hints) {
19         hint.tasks.remove(this);
20         hint.save();
21         hint.delete();
22     }
23     //Loesche die Aufgabe selbst
24     this.delete();
25 }
26 }
```

Listing 4.2: Auszug des Model-Objekts Task (Aufgabe)

Ebean legt für das Halten dieser Beziehungen sogenannte Mappingtabellen automatisch an (siehe Abbildung 4.1) und pflegt diese selbstständig.

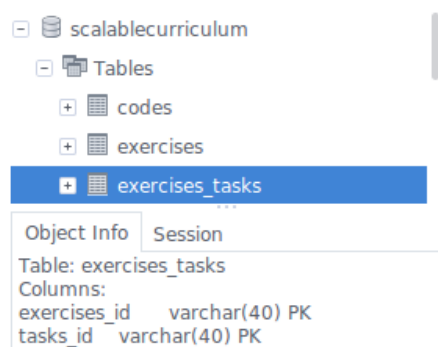


Abbildung 4.1: Mapping-Tabelle der Assoziationen (Übungen zu Aufgaben)

Man sollte bei Operationen, wie zum Beispiel »dem Löschen von Objekten« diese Beziehungen stets berücksichtigen und die Implementierung dementsprechend anpassen (siehe jeweils »public void del()« in den Listings 4.1 und 4.2), so-

4 Prototypische Implementierung der Webanwendung

dass keine »Leichen«⁹ entstehen können und in Auslese- bzw. Lade-Vorgängen Probleme entstehen. In Bezug auf Konsistenz und Stabilität, ist es sehr von Vorteil so weit wie möglich nicht manuell in die Datenbank einzugreifen, sondern stattdessen alles über das **ORM**-Framework abzuwickeln. Das eigenständige Anlegen von Structured Query Language (**SQL**)-Statements für das Abrufen der Geschäftsdaten der Anwendung, ist nicht notwendig. Das **ORM**-Framework Ebean stellt eine Menge an Lösungen und vorgefertigten Methoden bereit. Ein exemplarischer Quelltextauszug ist im folgenden Listing 4.3 zu sehen:

```
1 //Allg. Finder
2 public static Finder<String,User> find = new Finder<String,User>(
3   String.class, User.class
4 );
5
6 //gib User-Model anhand der Email - NULL bei nichtexistenz
7 public static User findByEmail(String email) {
8   return find.where().eq("email", email).findUnique();
9 }
```

Listing 4.3: Quelltextbeispiel für das Auslesen von Objekten

```
1 // Pruefe ob email noch nicht in der db registriert wurde
2 if(models.User.findByEmail(email) != null) {
3   return "E-Mail-Adresse bereits registriert!";
4 }
```

Listing 4.4: Quelltextbeispiel für den Aufruf von findByEmail()

Die Methode »findByEmail()« liefert bei Erfolg ein Userobjekt und bei Misserfolg *NULL*. Für die Implementierung wurde auf die von Ebean bereitgestellte Klasse »Model.Finder« mit den entsprechenden Methoden, welche letztendlich eigene **SQL**-Statements erzeugen, zurückgegriffen. Ein typisches Anwendungsbeispiel ist zum Beispiel im Listing 4.7 zu finden. Da als Anforderung bei der Entwicklung von »Scalable Curriculum« die Implementierung mit Java gilt, sind alle Modelklassen in Java programmiert. Wenn man sich in die Entwicklung mit Ebean¹⁰ eingearbeitet hat, ist ein effektiverer Arbeitsfluss möglich.

⁹ Als Leichen sind hierbei Datensätze in den Mappingtabellen gemeint, zu denen keine Objekte mehr in den eigentlichen Objekttabellen vorhanden sind.

¹⁰ siehe ebean-orm.github.io [9, Dokumentation]

4 Prototypische Implementierung der Webanwendung

Um Ebean als [ORM](#)-Framework dem Projekt zur Verfügung zu stellen, muss man das »Play Ebean Plugin« der »SBT Plugins«-Datei (project/plugins.sbt) hinzufügen:

```
1 addSbtPlugin("com.typesafe.sbt" % "sbt-play-ebean" % "3.0.0")
```

Listing 4.5: SBT Pluginregistrierung des Play Ebean Plugins

Anschließend muss man der »build.sbt« mitteilen, dass Ebean aktiv sein soll und beim Kompilieren des Projekts berücksichtigt wird:

```
1 lazy val root = (project in file(".")).enablePlugins(PlayJava, PlayEbean)
```

Listing 4.6: Play Ebean Plugin aktivieren

So lassen sich beliebig viele nützliche Plugins aktivieren und einbinden.

4.2.2 Controllers

Der gleiche Ansatz, bei der Programmierung auf Java zu setzen, gilt auch bei der Implementierung der Controller. Die Controller haben die Aufgabe, die in der Datenbank enthaltenen Models an die Benutzeroberfläche (Views) weiterzureichen, wenn diese vom Benutzer angefordert werden.

Dem gegenüber müssen die Controller auch die Steuerung und Aktionen des Benutzers, welche über die Benutzeroberfläche kommen, verarbeiten. Sie klassifizieren auch eingehende Daten in anwendungsspezifische Modelobjekte und stoßen eventuell die bereits erwähnte Speicherung in der Datenbank an. So ist die Kontrollschicht immer als Bindeglied zwischen den Models und den Views zu betrachten.

Ein Controller innerhalb einer Play Webanwendung sieht sehr übersichtlich aus. Analog zu den Modelklassen erben alle Controller von der Basisklasse

4 Prototypische Implementierung der Webanwendung

»play.mvc.Controller« und deklarieren eine beliebige Anzahl statischer Methoden, die sogenannten Controller-Actions (oder einfach Actions). Ein Controller muss mindestens eine Action implementieren. Den Actions können beliebig viele Parameter übergeben werden, welche automatisch gebunden werden. Sind hierbei primitive Datentypen (zum Beispiel »String«) übergeben worden, erfolgt eine automatische Bindung aus der Benutzeranfrage (Request) auf die Methodenparameter. In Ausnahmefällen kann die Bindung mit der Verwendung von Annotationen angepasst oder unterdrückt werden. Dies bietet sich insbesondere für Daten an, welche zum Beispiel sprachspezifisch formatiert sind oder aus Sicherheitsgründen gar nicht erst gebunden werden sollen. Einen beispielhaften Quelltextauszug eines Controllers sieht man im Listing 4.7:

```
1 [...]
2 public class Users extends Controller {
3     //uebergib komplette Userinstanz an die View
4     public static void show(String email) {
5         User user = User.findById(email);
6         render(user);
7     }
8     //stosse den Loeschvorgang im Usermodel an
9     public static void delete(String email) {
10        User user = User.findById(email);
11        user.del();
12    }
13 }
```

Listing 4.7: Quelltextbeispiel für den Aufruf von findById()

Listing 4.7 zeigt exemplarisch den Users-Controller, welcher die zwei Actions »show(String)« (zur Anzeige aller Benutzerdaten), sowie »delete(String)« (zum Löschen eines bestimmten Benutzers) anhand der Email-Adresse definiert. Die Action »show(String)« ruft, wie in Zeile 6 zu sehen, einen Renderprozess eines dem Controller zugewiesenen Templates auf und übergibt diesem eine User-Instanz, unter der Voraussetzung, dass die übergebene Emailadresse einem in der Datenbank gehalten Users zugewiesen ist. Das Template kann diese Instanz entgegennehmen und diese objekttypisch behandeln, indem es zum Beispiel Benutzerdaten, wie »Vorname«, »Nachname« oder »Emailadresse« abfragt und

4 Prototypische Implementierung der Webanwendung

diese an geeigneter Stelle ausgibt.

Allgemein hat der Benutzer — unter Verwendung des Webbrowsers — nicht die Möglichkeit einen Controller direkt anzusprechen. Der Weg zum Aufruf eines Controllers geht bei einer Webanwendung in der Regel über eine Route ([URL](#)). Diese Routen generieren sich nicht selbst, sondern müssen von dem Entwickler manuell definiert werden. Hierfür wird eine Konfigurationsdatei unter »/conf« namens »routes« angelegt:

```
1 [...]
2 # Allgemein
3 # *                /{controller}/{action}    {controller}.{action}
4 # Authentication Routing
5 GET                /login                    controllers.AuthenticationController.login
6 POST               /login                    controllers.AuthenticationController.authenticate
7 # Beispielrouten fuer Listing 4.7
8 GET                /user/daten/{email}       controllers.Users.show
9 GET                /group/{group_id}/member/{position} controllers.Group.showMember
10 [...]
```

Listing 4.8: Quelltextbeispiel für die Routing-Konfiguration

Wie im Listing 4.8 in Zeile 8 zu erkennen ist, werden bei Play die Zuweisungen von [HTTP](#)-Methoden und Uniform Resource Identifier ([URI](#))-Mustern zu Controller-Methoden (Actions) in folgender Syntax definiert: In der ersten Spalte wird jeweils die [HTTP](#)-Methode vermerkt. Hierbei sind alle Methoden, welche vom [HTTP](#)-Standard unterstützt werden zugelassen (GET, POST, PUT, DELETE). In der zweiten Spalte wird die [URI](#) festgehalten. Dem Entwickler sind bei der Bezeichnung der einzelnen Bestandteile einer URI keine Grenzen gesetzt. Diese sollten jedoch [HTTP](#)-konform sein und eventuell einen beschreibenden Charakter aufweisen. Innerhalb der [URI](#) sind auch dynamische Bestandteile — zur Realisierung der Parameterübergabe — möglich. Dynamische Bestandteile müssen, wie in Zeile 8 zu sehen ist, in geschweiften Klammern geschrieben werden. So ist es zum Beispiel auch möglich statische und dynamische Bestandteile in ihrer Reihenfolge miteinander zu vermischen (siehe hierzu Zeile 9). Das Routenmanagement in Play bringt noch einige weitere sehr komfortable und

hilfreiche Features mit, deren näheren Beschreibungen der Dokumentation¹¹ entnommen werden können.

4.2.3 Views

Die Benutzeroberfläche wird in Play mit der in Scala geschriebenen Template Engine Twirl¹² implementiert. Bei der Entwicklung von Twirl hat man sich stark an ASP.NET Razor orientiert. Twirl wird durch folgende Features beschrieben:

- **Kompakt und ausdruckstark:** Es verspricht durch sehr wenig Schreibaufwand eine hohe Effizienz in Bezug auf »coding workflow«. Im Vergleich zu anderen Template-Engines ist es mit Twirl nicht notwendig den dynamischen Codeteil mit einer extra Syntax in Form von Blöcken vom [HTML](#)-Code abzukapseln, da der Twirl-Parser diesen stets erkennt und interpretiert.
- **Schnell und einfach zu erlernen:** Die Lernkurve bei der erstmaligen Entwicklung mit Twirl ist sehr steil, da ausschließlich grundlegende Konstrukte aus der Scalawelt und gewöhnliche [HTML](#)-Konzepte verwendet werden.
- **Twirl ist keine neue Sprache:** Der Ansporn war es nicht mit Twirl eine neue Sprache zu schaffen, sondern den Scala-Entwicklern eine Möglichkeit zugeben, mit ihren Scala-Programmierkenntnissen eine dynamische Markup-Lösung in Verbindung mit [HTML](#) zu schaffen.
- **Programmieren in jedem Editor:** Ein sehr wichtiger Punkt ist, dass es sich mit Twirl in jeder [IDE](#) und jedem einfachen Editor (zum Beispiel vim, nano, etc.) entwickeln lässt. Diejenigen Entwickler, welche also mit Vim und co auf der Konsole programmieren, haben mit Twirl ein passendes Werkzeug zur Implementierung von Benutzeroberflächen in [HTML](#).

¹¹ siehe playframework.com [38, Dokumentation - HTTP Routing]

¹² siehe playframework.com [40, Dokumentation - The Template Enginge]

4 Prototypische Implementierung der Webanwendung

Bei der Implementierung der Benutzeroberflächen der Webanwendung »Scalable Curriculum« wurde großen Wert auf die Modularisierung der einzelnen Komponenten gelegt (siehe hierzu die passende Passage im Entwurf: Seite 44). Demnach stellen die einzelnen Komponenten Templates dar, welche je nach Bedürfnis verschachtelt werden können. So bildet die Datei »app/views/main.scala.html« ein Grundgerüst, welches das Template Navigation einbindet und diesem den Schlüssel »nav« übergibt. Das Navigationstemplate wertet diesen dann dementsprechend aus. Des weiteren erwartet das Maintemplate einen Parameter »content«, des Typs `HTML` oder `NULL`. Der `HTML`-Inhalt der Variable »content« wird im »body« ausgegeben. Das Maintemplate ist in Listing 4.9 auszugsweise zu sehen.

```
1 @* Projekt-Template: main *@
2 @(title: String, nav: String, user: User = null)(content: Html)
3 <!DOCTYPE html>
4 <html lang="de">
5   <head>
6     <title>@if(!title.isEmpty) { @title - } Scalable Curriculum</title>
7     <link rel="shortcut icon" type="image/png"
8       href="@routes.Assets.versioned("images/favicon.png") ">
9     [...]
10   </head>
11   <body >
12     <header>
13       <div id="sc-main-logo">
14         <a href="@routes.HomeController.index()" >
15           
16         </a>
17       </div>
18       @* Hier kommt die Navigation(param: schluessel): *@
19       @navigation(nav)
20     </header>
21     @* Hier kommt der uebergebene HTML Content: *@
22     @content
23   </body>
24 </html>
```

Listing 4.9: Auszug: app/views/main.scala.html

In dem Template der Navigation (siehe Listing 4.10) ist gut zu erkennen, wie einfach die zuvor erwähnten dynamischen Codesequenzen sich in den `HTML`-

4 Prototypische Implementierung der Webanwendung

Code einbauen lassen. Zeile 5 bis 8 zeigt eine if-Anweisung, welche den Authentifizierungsstatus des Benutzers abfragt. Auch ist der Scala/Java-typische Methodenaufruf, als »href-link« in Zeile 6, 7, 18, 22 und 24 gut zu erkennen.

```
1  @* Navigation-Template: navigation *@
2  @(nav: String)
3  <nav id="sc-main-navigation">
4    <ul>
5      @if(session.get("connected")){
6        <li class="@("active".when(nav == "dashboard"))"><a
7          href="@routes.DashboardController.index()">Dashboard</a></li>
8        <li class="@("active".when(nav == "editor"))"><a
9          href="@routes.EditorController.index()">Editor</a></li>
10      }
11    </ul>
12  </nav>
13  <nav id="sc-user-navigation">
14    <ul>
15      @if(session.get("connected")){
16        <li class="@("active".when(nav == "auth"))"><a class="auth"
17          onclick="false">@session.get("username")</a>
18        <ul class="dropdown">
19          <li><a>Einstellungen</a></li>
20          <li><a>Test</a></li>
21          <li><a href="@routes.AuthenticationController.logout()">Abmelden</a></li>
22        </ul>
23      } else {
24        <li class="@("active".when(nav == "auth"))"><a class="auth"
25          href="@routes.AuthenticationController.login()">Anmelden</a></li>
26      }
27    <li class="@("active".when(nav == "curriculum"))"><a
28      href="@routes.DashboardController.index()">Curriculum</a></li>
29    </ul>
30  </nav>
```

Listing 4.10: Auszug: app/views/navigation.scala.html

Die reinen HTML-Elemente sind noch durch CSS-Klassen visuell aufgewertet und bekommen erst durch diese ihr, für »Scalable Curriculum« typisches Erscheinungsbild. Die CSS-Klassen werden durch das, im Abschnitt 2.3 beschriebene, UIKit Framework bereitgestellt. UIKit muss im »public-Ordner« für das Projekt verfügbar gemacht werden (siehe hierzu im Anhang B.8) und

4 Prototypische Implementierung der Webanwendung

im »head« des auszugebenen [HTML](#)-Dokuments eingebunden werden¹³. Das Listing 4.11 zeigt die typische Verwendung von UIKit-Klassen für Elemente, wie zum Beispiel »Inputs«, »Buttons«, »Texte« oder »Layouts«:

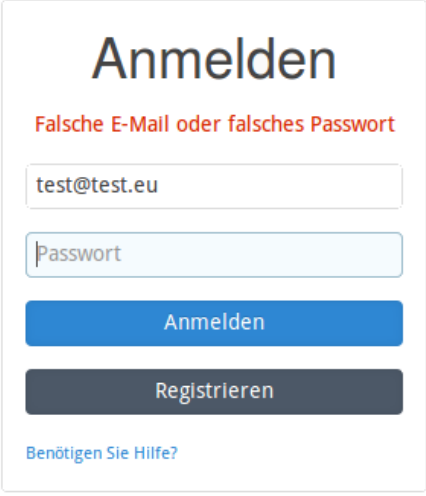
```
1 @(form: Form[classes.Login])
2 @main("Anmelden", "auth") {
3   <div class="uk-vertical-align uk-text-center sc-background-gradient">
4     <div class="uk-vertical-align-middle" style="width: 280px; margin-top: 20vh;">
5       @helper.form(routes.AuthenticationController.authenticate, 'class -> "uk-panel
6         uk-panel-box uk-form") {
7         <h1>Anmelden</h1>
7         @if(form.hasGlobalErrors) {
8           <p class="error">
9             <span class="uk-text-danger">@form.globalError.message</span>
10          </p>
11        }
12        [...]
13        <div class="uk-form-row">
14          <input class="uk-width-1-1" type="email" name="email"
15            placeholder="E-Mail-Adresse" value="@form("email").value">
16        </div>
17        [...]
18        <div class="uk-form-row">
19          <a class="uk-button sc-button-normal uk-width-1-1"
20            href="@routes.AuthenticationController.register()">Registrieren</a>
21        </div>
22        <div class="uk-form-row uk-text-left">
23          <a class="uk-text-small" href="">Benötigen Sie Hilfe?</a>
24        </div>
25        [...]
```

Listing 4.11: Auszug: app/views/login.scala.html

Ein gutes Beispiel für die Verwendung des UIKits zeigt insbesondere Zeile 9, in der ein Span-Element mit der Klasse »uk-text-danger« klassifiziert wird. Diese Klassifizierung bewirkt eine Ausgabe, wie sie in der Abbildung 4.2 zu sehen ist. Einige Elemente, wie zum Beispiel das Input-Element benötigen keine direkte Klassifizierung, um durch das UIKit ausgezeichnet zu werden. In diesen Fällen wird die [CSS](#)-Klassifizierung über Erbregelungen bestimmt.

¹³ Die UIKit-Klassen sind im Projekt »Scalable Curriculum« im »head« des [HTML](#)-Dokuments »main.scala.html« eingebunden.

4 Prototypische Implementierung der Webanwendung



The image shows a login form titled "Anmelden". It features a red error message "Falsche E-Mail oder falsches Passwort" above the email input field. The email field contains "test@test.eu". Below it is a password field labeled "Passwort". There are two buttons: a blue "Anmelden" button and a dark grey "Registrieren" button. At the bottom, there is a link "Benötigen Sie Hilfe?".

Abbildung 4.2: Ausgabe eines durch UIKit klassifizierten Fehlertextes

Für weiterführende Informationen und exemplarische Beispiele, ist in der UIKit-Dokumentation¹⁴ nachzuschlagen.

¹⁴ siehe getuikit.com [16, Dokumentation]

4.3 Grundlegende Klassen

4.3.1 Klasse: Runsc

Die Klasse »Runsc« (in Anlehnung an: run scala) implementiert das Kompilieren und Ausführen eines Scala-Codes. Diese Klasse findet in der Editor-Controller Anwendung. Der Prozess und die damit verbundene Arbeitsweise der Klasse »Runsc« lässt sich wie folgt beschreiben:

- **Quelltext absenden:** Über die Benutzeroberfläche wird der erarbeitete Scala-Quelltext (der Code zu einer Übung) über den Button »Kompilieren & Ausführen« (siehe Abbildung 4.3) via [HTTP-POST](#) an die Webanwendung übersandt. Durch das definierte Routing wird die Methode »compileCode()« des Editor-Controllers aufgerufen.

```
1 [...]
2 POST    /editor                                controllers.EditorController.compileCode
3 [...]
```

Listing 4.12: Routes-Konfiguration zu Kompilieren & Ausführen

- **Aufruf der »Runsc.execute()«:** Innerhalb der Action »compileCode« wird ein Objekt der Klasse »Runsc« initialisiert und »runsc.execute()« mit dem übergebenen Scala-Quelltext aufgerufen.

```
1 [...]
2 DynamicForm dynamicForm = Form.form().bindFromRequest();
3 String sScala = dynamicForm.get("code");\\
4 [...]
5 Runsc runsc = new Runsc();
6 this.results = runsc.execute(sScala);
7 [...]
```

Listing 4.13: Auszug des EditorControllers (compileCode-Action)

- **Quelltext Kompilieren:** Die erste Handlung der Methode »execute()« ist es den Quelltext in eine Datei »tmp/ScalaObject.scala« zu schreiben.

4 Prototypische Implementierung der Webanwendung

Nachdem die Datei erstellt wurde, wird der Scala-Compiler mit »runProcess("scalac -nowarn -d tmp "+ path)«, wobei »path« die Datei darstellt, aufgerufen. Die Methode »runProcess()« (siehe Listing 4.14) führt über »Runtime.getRuntime().exec(command)« ein Kommando aus und ruft, für jeweils einen InputStream und einen ErrorStream die Methode »println()« auf. Die Methode »println()« wiederum fügt der klasseninternen »static List<ProcessMessage> results«-Variable ein Item hinzu, welches die Rückgabe des zuvor ausgeführten Kommandos darstellt. Um diese Rückgabe besser und objektorientierter handhaben zu können, sind diese immer Objekte der Klasse »ProcessMessage« (siehe nächsten Abschnitt 4.3.2).

```
1 private static void runProcess(String command) throws Exception {
2     Process pro = Runtime.getRuntime().exec(command);
3     println(command, pro.getInputStream(), ProcessMessage.Status.NORMAL);
4     println(command, pro.getErrorStream(), ProcessMessage.Status.ERROR);
5     pro.waitFor();
6 }
```

Listing 4.14: Methode: runProcess()

- **»ScalaObject.scala« ausführen:** Nachdem die Quelltext-Datei erfolgreich kompiliert wurde, wird die Scala-Klasse mit »runProcess("scala -cp " + path.replace("ScalaObject.scala", "") + " ScalaObject");« ausgeführt. Da die Ausführung mit den selben Methoden, wie eben beschrieben erfolgt, sieht die Rückgabebewertung gleich aus.
- **Rückgabe der Liste:** Zuletzt antwortet die Methode »execute()« mit einer Liste »(List<ProcessMessage>)«.
- **Übergabe an die Editor-View:** Der Controller reicht die Liste der Ausgabezeilen direkt an die Editor-View weiter, welche diese in die Benutzeroberfläche integriert und dem Benutzer somit eine erfolgreiche Ausgabe präsentiert oder ihn auf eventuelle Kompilierungsfehler oder Ausführungsfehler hinweist (siehe Abbildung 4.4).

4 Prototypische Implementierung der Webanwendung

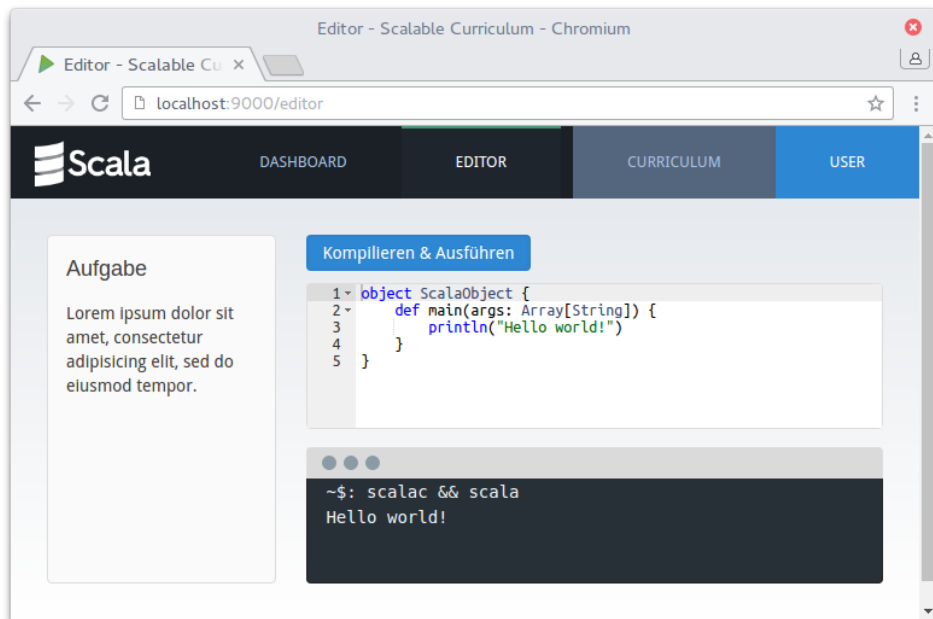


Abbildung 4.3: Editor-Benutzeroberfläche mit Quelltext und korrekter Ausgabe

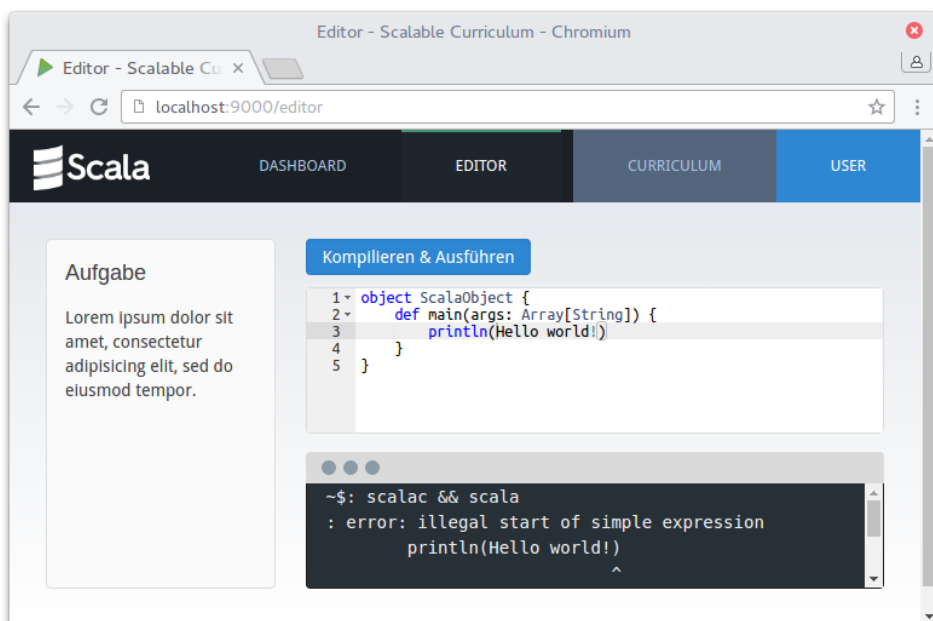


Abbildung 4.4: Die Editor-Benutzeroberfläche mit Quelltext und Fehlerausgabe

4.3.2 Klasse: `ProcessMessage`

Die Klasse »`ProcessMessage`« dient vordergründig der Klassifizierung von Ausführungsantworten, welche wie eben im Abschnitt 4.3.1 beschrieben, erzeugt werden, wenn die Ausführung von Code oder Kompilierungs- bzw. Laufzeitfehler-Ausgaben an die Konsole geliefert werden. Ein weiterer Vorteil, die Ausgabetexte als Objekte zu behandeln, ist die Möglichkeit der konkreteren Spezifizierung. So kann man mittels »`message.getStatus()`« abfragen, welche Art¹⁵ von Ausgabe diese gerade ist oder man lässt sich, wenn es zum Beispiel eine Fehlermeldung ist, mit »`getCommand()`« das Kommando, welches von »`runProcess()`« aufgerufen wurde, ausgeben. Das Listing 4.15 zeigt die wesentlichen Zeilen der »`ProcessMessage`« Klasse komprimiert:

```
1 package classes;
2 public class ProcessMessage {
3     public enum Status {
4         INPUT,
5         ERROR,
6         WARNING,
7         NORMAL,
8         UNDEF
9     }
10    private String message;
11    private Status status;
12    private String command;
13    public ProcessMessage(String message, Status status, String command) {
14        setMessage(message);
15        setStatus(status);
16        setCommand(command);
17    }
18    public void setStatus(Status status) {
19        if(status != null){
20            this.status = status;
21        } else {
22            this.status = Status.UNDEF;
23        }
24    }
25    public void setMessage(String message) {[...]
```

¹⁵ Die unterschiedlichen Arten, welche eine Ausgabe haben können, sind in einem Enum-Objekt gehalten. Hierzu gehören: »INPUT«, »ERROR«, »WARNING«, »NORMAL« und »UNDEF«.

4 Prototypische Implementierung der Webanwendung

```
26 public void setCommand(String command) {...}
27 public String getMessage() {...}
28 public Status getStatus() {...}
29 public String getCommand() {...}
30 [...]
```

Listing 4.15: Wesentlicher Quelltext der ProcessMessage Klasse

Das Halten der verschiedenen Ausgabearten — im Quelltext als »Status« deklariert — hat sich bereits bei der Implementierung der View als positiv herausgestellt, da eine Interpretation in Bezug auf den Kontext einer Ausgabe für die visuelle Gestaltung notwendig ist. In der Benutzeroberfläche sind so Ausgabetexte, bei denen die Methode »getStatus()« den Wert »ERROR« liefert, visuell *ROT* dargestellt.

5 Zusammenfassung

Dieses Kapitel beschreibt abschließend eine Zusammenfassung der entstandenen Ergebnisse, sowie deren Bewertung unter Berücksichtigung der im Kapitel 3 erarbeiteten Zielsetzung. Im letzten Abschnitt des Kapitels werden Anmerkungen und Vorschläge für zukünftige Erweiterungen und Optimierungen des Projekts »Scalable Curriculum« diskutiert.

5.1 Auswertung der Entwicklung

Das Ziel der Arbeit war die Entwicklung einer Webanwendung, die als E-Learning System die Möglichkeit geben soll die Programmiersprache Scala sowohl zu erlernen als auch zu lehren. Die Implementierung sollte unter Verwendung des Play Frameworks in Java realisiert werden. Dabei sollten alle Vorteile, welche bei der Entwicklung mit dem Play Framework existieren untersucht und bewertet werden. Unter diesen Aspekten, wurde in der vorliegenden Arbeit die Webanwendung »Scalable Curriculum« konzeptioniert und prototypisch implementiert. Hierbei wurden besonders die von Play vorgeschlagenen Konzepte stark berücksichtigt und angewandt. »Scalable Curriculum« stellt mit seiner stabilen Editor-Komponente, der Curriculumsverwaltung und dem gesamten Benutzermanagement eine stark modularisierte Software dar, was sich positiv auf die Skalierbarkeit, sowie Wartbarkeit der Software auswirkt. Die Editor-Komponente liefert, in Verbindung mit der Kursverwaltung dabei eine benutzerfreundliche und interaktive Möglichkeit, in Begleitung von informativen Hilfetexten, eingebettet in Kursen, mit Übungen und Aufgaben die

5 Zusammenfassung

Programmiersprache Scala *Schritt-für-Schritt* zu erlernen bzw. zu lehren. Unter der Voraussetzung bei der Entwicklung mit den Initialtemplates, welche die Software Activator von vornherein bereitstellt, und nicht bei *NULL* beginnen zu müssen, ist deutlich der Vorteil, eines sehr schnellen Einstiegs in die Entwicklung mit Play zu bekommen, bestätigt. Das bei der Entwicklung verwendete Initiatemplate (siehe Anhang C) wies bereits ein erstes vollständiges Grundgerüst, auf Basis des unter Abschnitt 2.1.2 beschriebenen MVC-Architekturmusters. So konnte man sich stets daran orientieren und besaß zugleich einen gewissen Leitfaden, wie Models, Controllers und Views implementiert sein müssen. So ist auch die Hot-Reload-Funktion und Errorlog im Webbrowser-Funktion ein sehr hilfreiches und komfortables Feature. Gerade in der Anfangszeit, wenn erstmalig mit Play oder Java entwickelt wird, stellen diese Funktionen einen großen Vorteil, gegenüber der Entwicklung mit anderen Systemen dar. Für das Verwalten der Objektdaten, wie zum Beispiel »Benutzer«, »Kurse«, »Übungen«, »Aufgaben« und »Hinweistexte« in einer Datenbank konnte der, von Play bereitgestellte Objekt Relation Mapper Ebean, durch seine einfache Handhabung und guter Dokumentationen überzeugen und fand bei »Scalable Curriculum« Anwendung. Im Hinblick auf die Entwicklung und Gestaltung der Benutzeroberflächen mit dem UIKit Framework ist vorallem die Kombinierbarkeit der einzelnen CSS-Klassen von Vorteil, wodurch sich durch die Kombination mehrerer CSS-Klassen schnell und effizient das gewünschte Layout realisieren lässt. Es ist jedoch ratsam stets einen Blick in die Dokumentation des Frameworks zu werfen, da dort bereits viele Beispiele verfügbar und ausreichend dokumentiert sind. Auch war die Entscheidung, vor Beginn der Implementierung der Benutzeroberflächen, mithilfe des UIKit-Customizers eine auf die visuellen Ansprüche abgestimmte Version des Frameworks zu erzeugen, vollkommen richtig. Dies hat eine nötige Korrektur der einzelnen Werte, welche das Erscheinungsbild zum Beispiel farblich beeinflussen, verhindert.

Die Verwendung der objektorientierten Programmiersprache Java war im Bereich der Webentwicklung, im direkten Vergleich zu Skriptsprachen, wie zum Beispiel PHP zunächst ein wenig umständlich, jedoch überzeugt sowohl die

5 Zusammenfassung

Typsicherheit, als auch die daraus resultierende Stabilität und Genauigkeit. In Verbindung mit dem Webframework Play ist eine typsichere, stabile und effektive Webentwicklung mit Java garantiert. Die daraus folgenden Vorteile werden mit steigender Komplexität der zu entwickelnden Webanwendung deutlicher. Dies bedeutet, dass für die Umsetzung eines kleinen Programmes eventuell eine Programmiersprache wie PHP oder Javascript vorzuziehen ist. Wie die Entwicklung des Projekts »Scalable Curriculum« gezeigt hat, sollte für große Software-Projekte, welche den Anspruch an Skalierbarkeit und Stabilität stellen das Play Framework durchaus die erste Wahl sein.

5.2 Ausblick

Die aus der Entwicklung resultierenden Funktionalitäten entsprechen im Wesentlichen denen der zuvor definierten Anforderungen, wobei bereits während der Implementierung neue Funktionen (wie beispielsweise die »Kontaktierung des Autors einer Übung«) hinzugekommen sind und somit spezielle Erweiterungen darstellen. Wenn die Webanwendung um eine weitere Funktion, zum Beispiel einer Kommentarfunktion, welche es den Benutzern ermöglicht Übungen zu kommentieren, erweitert wird, würde dies sowohl die Benutzerfreundlichkeit als auch die Attraktivität der Webanwendung steigern. Auch wäre es denkbar möglich die Übungen und Kurse zu bewerten oder zu favoritisieren. Durch die Gegebenheit, dass die Webanwendung sehr modular konzeptioniert ist, wäre auch — was jedoch nicht im Sinne dieser Arbeit ist — ein dynamischer Austausch der Klasse »Runsc«, welche für die Kompilierung und Ausführung des Scala-Codes verantwortlich ist, sehr einfach realisierbar. Dies würde als Folge haben, dass das E-Learning System »Scalable Curriculum« mehrere Programmiersprachen unterstützen könnte und demnach keine Beschränkung auf Scala hätte.

Die hier zusammengetragenen Ideen gelten als Ansatz für sinnvolle Erweiterungen der ursprünglichen, hier erarbeiteten Zielstellung und der unter Kapitel 3

5 Zusammenfassung

definierten Anforderungen. Es soll ausdrücklich erwähnt sein, dass die vorgestellten Ideen für mögliche Erweiterungen als Motivation und potentieller Anhaltspunkt einer zukünftigen Weiterentwicklung des Projekts dienen sollen.

Das Projekt ist im folgenden Repository verfügbar: »git.kutzmann.eu/ck/scalable-curriculum.git«

Anhang A

Installation und Inbetriebnahme der Webanwendung

A.1 Voraussetzungen für die Projektaufnahme und den Betrieb

Für die Projektaufnahme und den Betrieb der Webanwendung »Scalable Curriculum« ist zwingend das Java Development Kit (verwendet wurde: OpenJDK8¹) und der Scala Compiler² (z.B.: Vers.2.11.6) notwendig. Zudem wird folgende Software benötigt:

- Die Versionsverwaltungssoftware Git für das Initialisieren des Projekts bzw. das Kopieren des Quelltextes der Webanwendung aus dem Git-Repository: git.kutzmann.eu/ck/scalable-curriculum.git
- Ein MySQL-Server mit einer leeren Datenbank namens »scalablecurriculum«. Wobei hier ein Blick in die Konfigurationsdatei »conf/application.conf« (Zeile 348 bis Zeile 351) weiterführende Informationen liefert.

¹ siehe openjdk.java.net [33, Offizielle Webseite]

² siehe scala-lang.org [43, Download]

Es ist sehr empfehlenswert, den Activator, der mit der Projektinitialisierung via Git mitkommt, in dem jeweiligen System global verfügbar zu machen, ansonsten muss der absolute Pfad stets mitgeführt werden.

A.2 Anleitung zur Projektaufnahme und Inbetriebnahme

Unter der Bedingung, dass die unter [A.1](#) aufgelisteten Voraussetzungen erfüllt sind, kann das Projekt wie folgt aufgenommen und die Webanwendung ausgeführt werden:

- In dem Verzeichnis, unter dem der Projektordner geclost werden soll muss folgendes Kommando abgeschickt werden:
»git clone http://git.kutzmann.eu/ck/scalable-curriculum.git«.
Danach sollte das gesamte Projekt lokal, unter dem Ordner »scalablecurriculum« verfügbar sein.
- Nun sollte wie in unter [A.1](#) bereits erwähnten »conf/application.conf« die Einstellungen an das lokale System angepasst werden (hierzu zählen vorallem die Datenbankeinstellungen³).
- Nachdem alle Einstellungen angepasst sind kann die Webanwendung aus dem Projektordner mit »activator run« ausgeführt werden. Falls dabei Fehler bzw. Probleme entstehen sollten, werden diese meist sehr verständlich entweder in der Weboberfläche (erreichbar über »localhost:9000«) oder innerhalb der Konsole, von der man den Befehl abgesetzt hat, beschrieben.

³ Wenn kein MySQL-Server extra laufen soll, können die Werte für »db.default.driver=org.h2.Driver« und »db.default.url=jdbc:h2:mem:play« gesetzt werden; So läuft die Datenbank virtuell im Systemspeicher des Entwicklungsrechners, *solange* die Webanwendung ausgeführt wird.

Anhang B

Beschreibung der Verzeichnisstruktur

Dieser Abschnitt beschreibt die hierarchische Verzeichnisstruktur und den Inhalt des nach der unter [A.2](#) erläuterten Installationsanleitung angelegten Verzeichnisses »scalablecurriculum«, das den gesamten Quelltext der Webanwendung enthält. Die folgende Abbildung zeigt die Verzeichniswurzel:

```
scalablecurriculum/  
├─ app/  
├─ conf/  
├─ project/  
├─ public/  
├─ test/  
├─ activator  
├─ activator-launch-1.3.7.jar  
├─ build.sbt  
├─ LICENSE  
└─ README.md
```

Abbildung B.1: Verzeichnisbaum: scalablecurriculum/

Im Folgenden werden die Dateien und Verzeichnisse der Ebene näher beschrieben.

B.1 Das app/-Verzeichnis

Das app/-Verzeichnis enthält alle zu kompilierenden Quelldateien der Webanwendung (Application sources[playframework.com]).

```
scalablecurriculum/  
└─ app/  
    ├── assets/  
    │   ├── stylesheets/  
    │   └── javascripts/  
    ├── classes/  
    ├── controllers/  
    ├── models/  
    └── views/
```

Abbildung B.2: Verzeichnisbaum: app/

Mit dem assets/-Verzeichnis besteht die Möglichkeit sowohl Stylesheet-Dateien, im [LESS](#)-Format als auch Javascript-Dateien, welche in CoffeeScript geschrieben sind, dem Projekt zur Verfügung zu stellen. Diese werden dann beim Kompilieren des Projekts mit kompiliert. In »Scalable Curriculum« findet man jedoch weder [LESS](#) noch CoffeeScript, sodass diese leer bleiben.

B.2 Das models/-Verzeichnis

Das models/-Verzeichnis enthält alle unter Abschnitt 4.2.1 beschriebenen Klassen, welche die Objekte, sowie Geschäftslogiken der Webanwendung abbilden (Application business layer[playframework.com]).

```
scalablecurriculum/  
└─ app/  
    └─ models/  
        └─ Code.java  
        └─ Exercise.java  
        └─ Hint.java  
        └─ Task.java  
        └─ User.java
```

Abbildung B.3: Verzeichnisbaum: models/

B.3 Das views/-Verzeichnis

Das views/-Verzeichnis beinhaltet alle unter Abschnitt beschriebenen Templates der Webanwendung (Templates[playframework.com]).

```
scalablecurriculum/  
└─ app/  
    └─ views/  
        ├── dashboard.scala.html  
        ├── editor.scala.html  
        ├── index.scala.html  
        ├── login.scala.html  
        ├── main.scala.html  
        ├── navigation.scala.html  
        └── register.scala.html
```

Abbildung B.4: Verzeichnisbaum: views/

B.4 Das controllers/-Verzeichnis

Die unter Abschnitt 4.2.2 beschriebenen Controller-Klassen befinden sich in dem controllers/-Verzeichnis (Application controllers[playframework.com]).

```
scalablecurriculum/  
└─ app/  
    └─ controllers/  
        ├── AuthenticationController.java  
        ├── DashboardController.java  
        ├── EditorController.java  
        └── HomeController.java
```

Abbildung B.5: Verzeichnisbaum: controllers/

B.5 Das classes/-Verzeichnis

Das Verzeichnis »classes« ist eine Erweiterung des von Activator gelieferten Initialtemplates, das für zusätzliche Klassen, welche Logiken und Prozesse abbilden gedacht ist. So sind in diesem folgende Dateien zu finden:

```
scalablecurriculum/  
└─ app/  
    └─ classes/  
        ├── Login.java  
        ├── ProcessMessage.java  
        ├── Register.java  
        ├── Runsc.java  
        └── Secured.java
```

Abbildung B.6: Verzeichnisbaum: classes/

Klassen, die auf Dateiebene im classes/-Verzeichnis liegen und im Package »classes« hängen stellen keine Objekte im Sinne des [MVC](#)-Architekturmusters dar. Diese erben auch nicht von der Klasse Model (Ebean) und werden demnach auch nicht in der Datenbank gehalten. Sie sind als reine Arbeits- bzw. Prozessklassen gedacht.

B.6 Das conf/-Verzeichnis

Das conf/-Verzeichnis beinhaltet folgende Konfigurationsdateien (Configurations files and other non-compiled resources[playframework.com]):

```
scalablecurriculum/  
└─ conf/  
    └─ application.conf  
    └─ routes
```

Abbildung B.7: Verzeichnisbaum: conf/

Die Datei »application.conf« ist die unter [A.1](#) gemeinte Konfigurationsdatei, die während der Projektaufnahme an das eigene System anzupassen ist. In der Datei »routes« stehen die unter Abschnitt [4.2.2](#) beschriebenen [URIs](#) der Webanwendung.

B.7 Das project/-Verzeichnis

Im project/-Verzeichnis befinden sich die Konfigurationsdateien des SBTs¹(sbt configuration files[playframework.com]).

```
scalablecurriculum/  
└─ project/  
    └─ build.properties  
    └─ plugins.sbt
```

Abbildung B.8: Verzeichnisbaum: project/

In der Datei »plugins.sbt« muss zum Beispiel, wie in Abschnitt 4.2.1 beschrieben das Ebean Plugin aktiviert werden.

¹ Das Simple Build Tool ist ein Bestandteil der Programmiersprache Scala und stellt ein wichtiges Werkzeug zur Erzeugung notwendiger Bibliotheken dar (mehr Informationen unter scala-sbt.org [44]).

B.8 Das public/-Verzeichnis

Das public/-Verzeichnis beinhaltet alle Dateien, die öffentlich erreichbar sein müssen, wie zum Beispiel [CSS](#)-, Javascript- oder Bilddateien (Public assets [[playframework.com](#)]).

```
scalablecurriculum/  
└─ public/  
    └─ images/  
    └─ javascripts/  
    └─ stylesheets/
```

Abbildung B.9: Verzeichnisbaum: public/

B.9 Das test/-Verzeichnis

Im Verzeichnis test liegen alle Test-Klassen (source folder for unit or functional tests[playframework.com]).

```
scalablecurriculum/  
└─ test/  
    └─ ApplicationTest.java  
    └─ IntegrationTest.java
```

Abbildung B.10: Verzeichnisbaum: test/

Im Rahmen der Entwicklung von »Scalable Curriculum« sind keine Tests implementiert worden, sodass die beiden Klassen »ApplicationTest« und »IntegrationTest« ignoriert werden können.

B.10 Sonstige Ordner und Dateien

Unter dem Projektverzeichnis »scalablecurriculum« liegen noch einige noch nicht aufgezeigte Verzeichnisse bzw. Dateien. Zu diesen zählen insbesondere das Git Verzeichnis »git« und eine Git Konfigurationsdatei »gitignore«, die alle nicht zu versionierenden Dateien listet. Auch entstehen nach der erstmaligen Ausführung bzw. Kompilierung der Webanwendung einige neue Verzeichnisse und Dateien. Der folgende Verzeichnisbaum soll einen groben Überblick über diese Dateien geben und dabei bewusst die zuvor aufgeführten Verzeichnisse und Ordner ausblenden:

```
scalablecurriculum/  
├── [...]  
├── target/  
├── logs/  
├── tmp/  
├── .git/  
├── .sbtserver/  
├── .classpath  
├── .gitignore  
└── [...]
```

Abbildung B.11: Verzeichnisbaum: scalablecurriculum/ (Sonstige)

Unterhalb des target/-Verzeichnisses (Generated stuff[playframework.com]) liegen alle während des Kompilierens generierten Dateien, wie zum Beispiel »Compiled web assets« oder »Compiled class files«. Im Verzeichnis »logs« befindet sich die Datei »application.log«, welche die Standardlogdatei darstellt (Default log file[playframework.com]).

»Scalable Curriculum« benötigt zur Kompilierung und Ausführung von Scala-Code ein Verzeichnis namens »tmp« unterhalb der Projektwurzel. Falls dieses Verzeichnis nicht existiert, weist ein Hinweistext, während der Ausführung auf das Fehlen des Verzeichnisses hin.

Anhang C

Initailtemplate Play Framework

C.1 Beispielhaftes Java Initialtemplate des Play Frameworks

Der folgende Verzeichnisbaum gibt einen Überblick über die Verzeichnisstruktur eines initialen Java Templates, welches mit dem Programm Activator via »activator new projektname play-java« erzeugt wurde:

```
projektname/
├── app/
│   ├── controllers/
│   │   ├── AsyncController.java
│   │   ├── CountController.java
│   │   └── HomeController.java
│   ├── filters/
│   │   └── ExampleFilter.java
│   ├── views/
│   │   ├── index.scala.html
│   │   └── main.scala.html
│   ├── Filters.java
│   └── Module.java
```


Anhang C Initailtemplate Play Framework

```
├── conf/
│   ├── application.conf
│   ├── logback.xml
│   └── routes
├── project/
│   ├── build.properties
│   └── plugins.sbt
├── public/
│   ├── images/
│   ├── javascript/
│   └── stylesheets/
├── test/
│   ├── ApplicationTest.java
│   └── IntegrationTest.java
├── build.sbt
├── LICENSE
└── README
```

Literaturverzeichnis

- [1] ACE.C9.IO: *Ace - The High Performance Code Editor for the Web*. Version: 2016. Quelle: <https://ace.c9.io>, Abruf: 15.09.2016
- [2] ARCHLINUX.DE: *KISS-Prinzip*. Version: 2016. <https://wiki.archlinux.de/title/KISS-Prinzip>, Abruf: 19.09.2016
- [3] ATOM.IO: *Atom Packages*. Version: 2016. <https://atom.io/packages/>, Abruf: 15.09.2016
- [4] BEUTH-HOCHSCHULE.DE: L. Piepmeyer: *Funktionale Programmierung - Das Typsystem von Scala*. Version: 2016. <http://public.beuth-hochschule.de/~knabe/fach/scala/11w-ats5/unterricht/09-Typsystem-von-Scala.pdf>, Abruf: 15.09.2016
- [5] BLUEJ.ORG: *A free Java Development Environment designed for beginners*. Version: 2016. <http://www.bluej.org/>, Abruf: 15.09.2016
- [6] BUYYA, Rajkumar ; SELVI, S T. ; CHU, Xingchen: *OBJECT ORIENTED PROG WITH JAVA*. ISBN 978-1-25908-325-9 : Tata McGraw-Hill, 2009
- [7] COMPUTERWOCHE.DE: Was *Java-Web-Frameworks* leisten. Version: 2014. <http://www.computerwoche.de/a/was-java-web-frameworks-leisten,3065497>, Abruf: 15.09.2016
- [8] DEREKYU.COM: *Abbildung - gmtut-008.png*. Version: 2007. <http://www.derekyu.com/tigs/forums/tutorials/gmtut/gmtut-008.png>, Abruf: 16.09.2016

Literaturverzeichnis

- [9] EBEAN-ORM.GITHUB.IO: *Ebean ORM - Documentation*. Version: 2016. <http://ebean-orm.github.io/docs>, Abruf: 15.09.2016
- [10] EBEAN-ORM.GITHUB.IO: *Ebean ORM for Java/Kotlin*. Version: 2016. <http://ebean-orm.github.io/>, Abruf: 15.09.2016
- [11] FH-WEDEL.DE: *Abbildung - bild1.jpg*. Version: 2016. <http://www.fh-wedel.de/~si/seminare/ws08/Ausarbeitung/06.vvw/img/bild1.jpg>, Abruf: 16.09.2016
- [12] FH-WEDEL.DE: *Abbildung - bild2.jpg*. Version: 2016. <http://www.fh-wedel.de/~si/seminare/ws08/Ausarbeitung/06.vvw/img/bild2.jpg>, Abruf: 16.09.2016
- [13] FH-WEDEL.DE: *Verteilte Versionsverwaltung*. Version: 2016. <http://www.fh-wedel.de/~si/seminare/ws08/Ausarbeitung/06.vvw/vvw1.htm>, Abruf: 15.09.2016
- [14] GETUIKIT.COM: *A Lightweight And Modular Front-End Framework*. Version: 2016. <http://getuikit.com/>, Abruf: 15.09.2016
- [15] GETUIKIT.COM: *UIKit - Customizer*. Version: 2016. <http://getuikit.com/docs/customizer.html>, Abruf: 15.09.2016
- [16] GETUIKIT.COM: *UIKit Documentation - Base*. Version: 2016. <http://getuikit.com/docs/base.html>, Abruf: 15.09.2016
- [17] GETUIKIT.COM: *UIKit Documentation - Core*. Version: 2016. <http://getuikit.com/docs/core.html>, Abruf: 15.09.2016
- [18] GIT-SCM.COM: *Abbildung - deltas.png*. Version: 2016. <https://git-scm.com/book/en/v2/book/01-introduction/images/deltas.png>, Abruf: 16.09.2016
- [19] GITHUB.COM: *UIKit - Releases*. Version: 2013. <https://github.com/uikit/uikit/releases/tag/v1.0.0>, Abruf: 15.09.2016
- [20] GITHUB.COM: *GitHub - How people build software*. Version: 2016. <https://github.com/>, Abruf: 15.09.2016

Literaturverzeichnis

- [21] GITHUB.COM: *Github Playframework People*. Version: 2016. <https://github.com/orgs/playframework/people>, Abruf: 15.09.2016
- [22] GOGS.IO: *Gogs - Documentation*. Version: 2016. <https://gogs.io/docs>, Abruf: 15.09.2016
- [23] GOGS.IO: *Gogs - Go Git Service - a painless self-hosted Git service*. Version: 2016. <https://gogs.io/>, Abruf: 15.09.2016
- [24] KENAI.COM: *Abbildung - java-compile.gif*. Version: 2016. https://kenai.com/attachments/wiki_images/chessgame/java-compile.gif, Abruf: 16.09.2016
- [25] LESSCSS.ORG: *LESS Language*. Version: 2016. <http://lesscss.org/>, Abruf: 15.09.2016
- [26] LIGHTBEND.COM: *Lightbend Activator - Templates*. Version: 2016. <https://www.lightbend.com/activator/templates>, Abruf: 15.09.2016
- [27] LIGHTBEND.COM: *Lightbends Activator*. Version: 2016. <https://www.lightbend.com/community/core-tools/activator-and-sbt>, Abruf: 15.09.2016
- [28] LIGHTBEND.COM: *Typesafe Changes Name to Lightbend*. Version: 2016. <https://www.lightbend.com/blog/typesafe-changes-name-to-lightbend>, Abruf: 15.09.2016
- [29] MECHITTECHNOLOGIES.COM: *Abbildung - snapshots.png*. Version: 2016. <https://git-scm.com/book/en/v2/book/01-introduction/images/snapshots.png>, Abruf: 16.09.2016
- [30] MECHITTECHNOLOGIES.COM: *Abbildung - Web-Application-Development.jpg*. Version: 2016. http://www.mechittechnologies.com/images/technology/Web_Application_Development.jpg, Abruf: 16.09.2016
- [31] MOOCK.ORG: *Abbildung - mvc-02.png*. Version: 2016. <http://www.moock.org/lectures/mvc/images/mvc-02.jpg>, Abruf: 16.09.2016

Literaturverzeichnis

- [32] OPEN SOURCE INITIATIVE: *The MIT License (MIT)*. Version: 2016. <https://opensource.org/licenses/MIT>, Abruf: 18.09.2016
- [33] OPENJDK.JAVA.NET: *OpenJDK*. Version: 2016. <http://openjdk.java.net>, Abruf: 15.09.2016
- [34] ORACLE.COM: *J2SE 5.0 in a Nutshell*. Version: 2004. <http://www.oracle.com/technetwork/articles/javase/j2se15-141062.html>, Abruf: 15.09.2016
- [35] PIEPMAYER, Lothar: *Grundkurs funktionale Programmierung mit Scala*. ISBN 978-3-446-42092-2 : Carl Hanser Verlag München Wien, 2010
- [36] PLAYFRAMEWORK.COM: *Anatomy of a Play application*. Version: 2016. <https://www.playframework.com/documentation/2.5.x/Anatomy>, Abruf: 15.09.2016
- [37] PLAYFRAMEWORK.COM: *The High Velocity Web Framework For Java and Scala*. Version: 2016. <https://www.playframework.com/>, Abruf: 15.09.2016
- [38] PLAYFRAMEWORK.COM: *HTTP Routing*. Version: 2016. <https://playframework.com/documentation/1.1/routes>, Abruf: 15.09.2016
- [39] PLAYFRAMEWORK.COM: *The Main Concepts*. Version: 2016. <https://www.playframework.com/documentation/1.0/main>, Abruf: 15.09.2016
- [40] PLAYFRAMEWORK.COM: *The Template Engine*. Version: 2016. <https://www.playframework.com/documentation/2.5.x/JavaTemplates>, Abruf: 15.09.2016
- [41] SCALA-LANG.ORG: *Official Siemens SIS Press Release about ESME released*. Version: 2009. <http://www.scala-lang.org/old/node/1158.html>, Abruf: 15.09.2016
- [42] SCALA-LANG.ORG: *Novell Pulse - Lift at heart*. Version: 2010. <http://www.scala-lang.org/old/node/6618>, Abruf: 15.09.2016

Literaturverzeichnis

- [43] SCALA-LANG.ORG: *Download | The Scala Programming Language*. Version: 2016. <http://www.scala-lang.org/download/>, Abruf: 15.09.2016
- [44] SCALA-SBT.ORG: *SBT - The Interactive Build Tool*. Version: 2016. <http://www.scala-sbt.org/>, Abruf: 15.09.2016
- [45] SELFHTML.ORG: *Webstandards/responsive Webdesign*. Version: 2016. https://wiki.selfhtml.org/wiki/Webstandards/responsive_Webdesign, Abruf: 19.09.2016
- [46] SOURCEFORGE.NET: *SourceForge - Download, Develop and Publish Free Open Source*. Version: 2016. <https://sourceforge.net/>, Abruf: 15.09.2016
- [47] STACHMANN, Björn ; PREISSEL, René: *Git: Dezentrale Versionsverwaltung im Team Grundlagen und Workflows*. ISBN 978-3-86490-311-3 : Dpunkt.verlag GmbH, 2015
- [48] T3N.DE: *31 CSS-Frontend-Frameworks im Vergleich*. Version: 2013. <http://t3n.de/news/31-css-frontend-frameworks-450474/>, Abruf: 19.09.2016
- [49] TEIALEHRBUCH.DE: *Java - Geschichte und Ausblick*. Version: 2016. <https://www.teialehrbuch.de/Kostenlose-Kurse/JAVA/6528-Java-Geschichte-und-Ausblick.html>, Abruf: 15.09.2016
- [50] THESTYLEWORKS.DE: *CSS-Vererbungsregeln zusammengefasst*. Version: 2016. <http://www.thestyleworks.de/basics/inheritance.shtml>, Abruf: 15.09.2016
- [51] TWITTER.GITHUB.IO: *Effective Scala*. Version: 2012. <http://twitter.github.io/effectivescala/>, Abruf: 15.09.2016
- [52] ULLENBOOM, Christian: *Java ist auch eine Insel*. ISBN 978-3-8362-1506-0 : Rheinwerk Computing, 2010

Literaturverzeichnis

- [53] UNI-BONN.DE: *Sebastian Mancke's CVS-Einführung*. Version: 2016. <http://www.iai.uni-bonn.de/III/lehre/AG/OOP/material/cvs-show.pdf>, Abruf: 15.09.2016
- [54] UNI-MANNHEIM.DE: *Java, Packages*. Version: 2006. <http://krum.rz.uni-mannheim.de/pk1/sess-606.html>, Abruf: 15.09.2016
- [55] UNI-MANNHEIM.DE: *CGI Common Gateway Interface*. Version: 2016. <http://krum.rz.uni-mannheim.de/cgi-tut.html>, Abruf: 15.09.2016
- [56] YOOTHEME.COM: *YOOtheme GmbH*. Version: 2016. <https://yootheme.com/>, Abruf: 15.09.2016