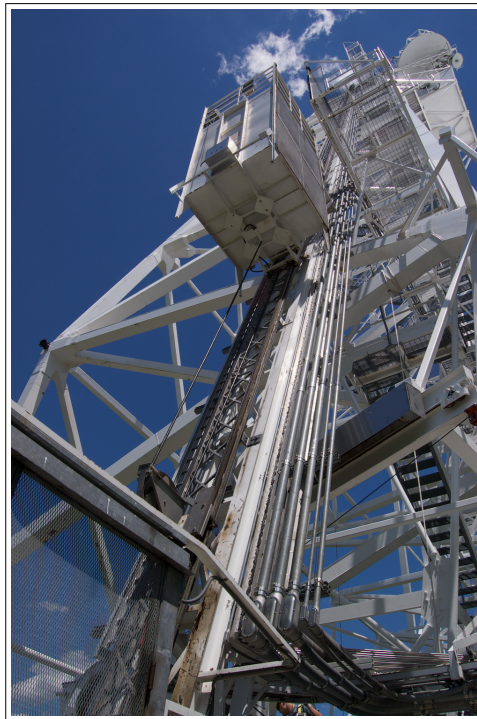


Entwicklung von Web-Applikationen mit Lift und Scala

Einführung anhand einer durchgehenden Beispielapplikation



Thomas Fiedler
Christoph Knabe

11. Januar 2011

Thomas Fiedler studierte bis 2010 Medieninformatik an der Beuth-Hochschule für Technik Berlin. Er arbeitet zur Zeit als freiberuflicher Softwareentwickler im Bereich der Web- und Oberflächenprogrammierung.



Christoph Knabe ist seit 1990 Professor für Softwaretechnik und Programmierung an der Beuth-Hochschule für Technik Berlin und übt für den Fachbereich VI die Funktion des Web-Beauftragten aus. Seine Interessensgebiete sind Scala, Lift, Web-Programmierung, AspectJ, Ausnahmebehandlung und Software-Qualität. Er unterrichtet im Studiengang Medieninformatik.



<http://public.beuth-hochschule.de/~knabe/>

Titelbild: Lift am Green Bank Telescope, Pocahontas County, West Virginia.

Fotograf: Alan Aversa.

http://commons.wikimedia.org/wiki/File:GBT_Elevator.png

Inhaltsverzeichnis

1. Einleitung	1
1.1. Aufbau des Buches	1
1.2. Grundsätzliches	3
1.3. Verwendung der Beispielanwendung	3
1.4. Typographische Hervorhebungen	4
2. Scala	5
2.1. Eigenschaften	5
2.1.1. Objektorientierung	5
2.1.2. Funktionale Programmierung	6
2.1.3. Erweiterbarkeit	6
2.1.4. Statisches Typsystem	6
2.1.5. XML-Unterstützung	7
2.2. Entwicklungsumgebungen	7
2.3. Leistung	8
3. Lift	9
3.1. Vorbilder	9
3.2. Nutzung von Scala-Features	10
4. Maven und Lift	11
4.1. Kurzbeschreibung	11
4.2. Konfiguration	11
4.3. Repositories	12
4.4. Plugins	12
4.5. Die POM-Datei	13
4.6. Einsatz in Lift	14
4.6.1. Erstellung eines neuen Lift-Projekts	14
4.6.2. Kompilierung	15
4.6.3. Deployment und Inbetriebnahme	15
4.6.4. Bearbeitung des Projekts für die Nutzung in einer IDE	17
4.6.5. Erstellung einer Quelltext-Dokumentation	18

5. Die Beispielanwendung „Lehrkraftnews“	19
5.1. Kurzbeschreibung	19
5.1.1. Benutzerverwaltung	19
5.1.2. Nachrichten	20
5.1.3. Abonnements	20
5.1.4. Zugrunde liegendes Datenmodell	20
5.1.5. Besonderheiten	21
5.2. Installation und Inbetriebnahme	22
5.2.1. Installation der benötigten Software	23
5.2.2. Inbetriebnahme der Anwendung	25
5.2.3. Bedienung	29
6. Test-Infrastruktur und Testorientierte Dokumentation	31
6.1. Test-Infrastruktur	31
6.1.1. Framework-Auswahl, JUnit, SUnit, specs	31
6.1.2. ScalaTest	32
6.1.3. Nutzung von ScalaTest/JUnit in Maven und der IDE	33
6.2. Testgetriebene Entwicklung	33
6.3. Testorientierte Dokumentation	35
6.3.1. Extraktion von Codeschnipseln	35
6.3.2. Testfälle als Programmbeispiele	35
7. Das Mapper-Framework	37
7.1. Grundsätzliches	37
7.1.1. Mapper/Record	37
7.2. Integration des Mapper-Moduls in das Projekt	37
7.3. Konfiguration des Datenbankzugriffs	38
7.4. Erstellung von Modell-Klassen	41
7.4.1. Mapper- und MetaMapper-Trait	45
7.4.2. Primärschlüssel mit LongKeyedMapper und IdPK	45
7.4.3. Festlegung des Tabellennamens	45
7.4.4. Definition von Datenfeldern	46
7.4.5. Hinzufügen von Hilfsfunktionen	46
7.4.6. Validierung von Datenfeldern	47
7.5. Objekt-Operationen	48
7.5.1. Erstellen und Speichern	48
7.5.2. Zugriff auf Datenfelder	49
7.5.3. Zugriff auf Objekt-Verbindungen	50
7.5.4. Aktualisieren	50
7.6. Datenbankabfragen	51
7.6.1. Methoden	51
7.6.2. Vergleichs-Query-Parameter	52

7.6.3.	Steuer-Query-Parameter	54
7.6.4.	Native SQL-Statements	55
7.6.5.	Zusammenfassung	57
7.7.	Erzeugung des Datenbankschemas mit Schemifier	57
7.8.	CRUD-Funktionalität durch den CRUDify-Trait	58
7.9.	ProtoUser und MegaProtoUser	61
7.9.1.	ProtoUser	62
7.9.2.	MegaProtoUser und MetaMegaProtoUser	63
8.	Views und Templates	71
8.1.	View First-Pattern	71
8.2.	Arbeit mit Templates	72
8.2.1.	Pfadstruktur und Benennung	72
8.2.2.	surround/bind-Tag	73
8.2.3.	embed-Tag	75
8.2.4.	ignore-Tag	75
8.2.5.	"templates-hidden"-Ordner	75
8.2.6.	Aufruf von Snippet-Methoden	76
8.2.7.	Einbettung von CometActors	76
8.2.8.	Internationalisierung mit Templates	77
8.2.9.	Head Merging	77
8.3.	Arbeit mit Views	77
8.3.1.	LiftView	77
8.3.2.	InsecureLiftView	81
8.4.	Ausgabe von Hinweisen, Warnungen und Fehlermeldungen	82
9.	Snippets	85
9.1.	bind	87
9.1.1.	Extraktion eines Parameters aus der URI	87
9.1.2.	Ersetzen von Platzhaltern durch bind	88
9.1.3.	Iteration und bind	89
9.1.4.	bind und chooseTemplate	90
9.2.	Formulare	92
9.3.	RequestVar	93
9.4.	SessionVar	97
9.5.	Stateful Snippets	97
10.	AJAX und Comet	103
10.1.	AJAX	103
10.2.	Comet	106
10.2.1.	Autonome CometActors	107
10.2.2.	Koordination von CometActors	116

11. Die Boot-Klasse	121
11.1. LiftRules	121
11.2. Mailer-Konfiguration	123
11.3. SiteMap	125
11.3.1. Erweiterte Zugriffssteuerung	126
11.3.2. Die SiteMap-DSL ab Lift 2	127
11.3.3. Generierung von Menüs	131
11.3.4. Festlegung des zu verwendenden Templates	135
11.3.5. Übersicht der Menüstruktur der Beispielanwendung	137
11.4. URL-Rewriting	138
11.5. Konfiguration des Datenbankzugriffs	140
11.6. Schemifier	140
11.7. Initialisierung von Widget-Klassen	140
12. Ausnahmebehandlung und Zentrales Ausnahm melden	141
12.1. Ausnahmebehandlung in der Anwendungssoftware	141
12.2. Zentrales Ausnahm melden bei Full-Page-Requests	142
12.3. Zentrales Ausnahm melden bei Ajax-Requests	146
12.4. Zentrales Ausnahm melden bei Comet-Requests	151
12.5. Ausnahm melden bei Hintergrund-Actors	152
12.6. Zusammenfassung	154
13. Internationalisierung	157
13.1. Internationalisierte Templates	157
13.2. properties-Dateien	157
13.2.1. lift-core.properties	157
13.2.2. lift.properties	158
14. Fazit	161
A. Anhang	163
A.1. Bearbeitung der Beispielapplikation mit einer IDE	163
A.2. Logging in der Applikation mit SLF4J über Log4J	163
A.3. Box, Full und Empty	165
A.4. Paginierung	166
A.5. Das TableSorter-Widget	169
A.6. Scala-Actors	172
A.7. Das S-Objekt	175
A.8. CometMailer	176
Literatur-, Abbildungs- und Programmauszugs-Verzeichnisse	181

1. Einleitung

Für die Entwicklung dynamischer Web-Anwendungen existiert eine Fülle verschiedener Webframeworks, die in den unterschiedlichsten Programmiersprachen implementiert sind. So vielfältig die Auswahl der Frameworks ist, so unterschiedlich sind ihre Ausprägungen und die Philosophien, die sie verfolgen. Bei der Entwicklung von *Lift* wurde versucht, die fortschrittlichsten Eigenschaften bestehender Frameworks zu vereinen und mit eigenen Ideen anzureichern. Gleichzeitig sollten Fehler bestehender Frameworks vermieden werden, die zur Erschwerung des Entwicklungsprozesses von Webanwendungen beitragen. *Lift* ist in der Programmiersprache *Scala* implementiert, deren moderne Konzepte bei der Umsetzung des Frameworks eine maßgebliche Rolle spielen [ChDW09].

Ziel dieses Buchs ist es, die unterschiedlichen Aspekte der Erstellung von Web-Anwendungen mit *Lift* zu beleuchten. Anhand der Beispielanwendung „Lehrkraftnews“ soll detailliert und konkret die Umsetzung von aus der Web-Entwicklung bekannten Aufgabenstellungen beschrieben werden. Dadurch soll ein Einstieg in die Entwicklung mit *Lift* ermöglicht, und über Vorzüge und Nachteile des Frameworks informiert werden. Aufgrund der mangelnden Dokumentation des Frameworks zum Zeitpunkt der Erstellung dieses Buchs wurde großer Wert darauf gelegt, möglichst viele in *Lift* zur Verfügung stehende Möglichkeiten anhand nachvollziehbarer Beispiele darzulegen. Anhand der gesammelten Erfahrungen soll eine Einschätzung über die Verwendbarkeit von *Lift* gegeben werden.

1.1. Aufbau des Buches

Kapitel 2 gibt zunächst einen Überblick über die wesentlichen Merkmale der Programmiersprache *Scala*. Eine Einführung in die Programmierung mit *Scala* soll an dieser Stelle jedoch nicht erfolgen (siehe Abschnitt 1.2).

Kapitel 3 informiert kurz über die zugrunde liegenden Konzepte des Frameworks *Lift* und seine Vorbilder.

Die Verwendung des Build-Management-Tools *Maven*, welches bei der Erstellung und Verwaltung von *Lift*-Projekten eine wichtige Rolle spielt, wird in Kapitel 4 beschrieben.

Kapitel 5 dient der Beschreibung der Beispielanwendung „Lehrkraftnews“. Nach einem Überblick über die Funktionsweise der Anwendung werden die nötigen Schritte für deren Installation und Inbetriebnahme erklärt.

In Kapitel 6 wird besprochen, wie man eine Infrastruktur für testgesteuerte Entwicklung aufbaut und wie diese zusammen mit einem Codeschnipsel-Extraktor bei der Erstellung dieses Buches genutzt wurde, um durchgängig korrekte und aktuelle Programmbeispiele und -auszüge abzudrucken.

Die Kapitel 7 bis 13 befassen sich konkret mit der Umsetzung von Web-Anwendungen mit Lift: Kapitel 7 behandelt die Nutzung des *Mapper*-Frameworks für die objekt-relationale Abbildung eines Datenmodells. Darüber hinaus stellt das Mapper-Framework Funktionalitäten für eine Benutzerverwaltung und für die Generierung sogenannter *CRUD*-Seiten bereit, deren Nutzung und Modifikation dort ausführlich besprochen werden.

Kapitel 8 beschäftigt sich mit den Möglichkeiten der Darstellung von Lift-Anwendungen. Nach einer kurzen Erklärung des *View-First*-Patterns wird die Nutzbarkeit von *Templates* und *View*-Klassen erläutert.

Die Programmsteuerung durch *Snippets* wird in Kapitel 9 beschrieben. Dabei wird auf das Zusammenspiel von *Templates* und *Snippets* eingegangen und die Erstellung und Auswertung von Formularen erklärt. Außerdem werden verschiedene Wege aufgezeigt, um zustand-behaftete Anwendungen zu erstellen.

In Kapitel 10 werden die Konzepte asynchronen Datentransfers erläutert, die unter den Begriffen *AJAX* und *Comet* zusammengefasst werden. Es folgt eine Beschreibung der Umsetzung dieser Konzepte in Lift.

Kapitel 11 befasst sich mit der Klasse *Boot* und deren zentraler Rolle für die Konfiguration von Lift-Anwendungen. Darunter fallen neben der Wahl des Datenbank-Treibers und des SMTP-Servers für den Versand von E-Mails auch die Definition einer Grundstruktur der Anwendung, anhand derer eine Zugriffssteuerung und die Generierung von Menüs erfolgen kann.

Eine Strategie zur *Ausnahmebehandlung* wird in Kapitel 12 beschrieben. Mittels Lift-spezifischer Eingriffspunkte konnte so *Zentrales Ausnahmemelden* in der Beispielapplikation durchgesetzt und robustes Verhalten erreicht werden.

Kapitel 13 widmet sich der mehrsprachigen Gestaltung von Lift-Anwendungen. Es werden die zwei grundlegenden Konzepte für die Internationalisierung vorgestellt.

In Kapitel 14 wird ein Fazit über die gesammelten Erfahrungen bei der Web-Entwicklung mit Lift gezogen. Zusätzlich soll an dieser Stelle ein kurzer Ausblick auf die Zukunft von Lift erfolgen.

Im Anhang ab Seite 163 werden Sachverhalte besprochen, die zum Verständnis von Lift bzw. der Beispielanwendung einer Erklärung bedürfen. Zur Verbesserung der Lesbarkeit dieses Buches werden sie gesondert behandelt, an geeigneten Stellen existieren Verweise auf die Themen im Anhang.

1.2. Grundsätzliches

Auf eine gesonderte Einführung in die Programmierung mit Scala wird im Rahmen dieses Buchs verzichtet, um dem Thema des Buchs durch den Fokus auf das Framework gerecht werden zu können. Stattdessen werden Programmauszüge an Ort und Stelle umfangreich erklärt, um ihre Auswirkungen für den Leser nachvollziehbar zu machen. Für die eigenständige Arbeit mit Lift ist eine umfangreiche Einarbeitung in die Programmiersprache Scala jedoch unumgänglich.¹

Die Versionen von Scala und Lift, mit denen die diesem Buch zu Grunde gelegte Beispielanwendung entwickelt wurde, sind im folgenden Dateiausschnitt angegeben.

```
!!-- Datei pom.xml, Ausschnitt versions: --->
<scala.version>2.8.0</scala.version>
<lift.version>2.2</lift.version>
```

Dabei handelt es sich um die bei der Erstellung dieses Buchs aktuellsten stabilen Versionen. Sofern nicht anders angegeben, beziehen sich auch alle Texte in diesem Buch auf die angegebene Lift-Version.

1.3. Verwendung der Beispielanwendung

Die Beispielanwendung liegt in einem *git*-Repository bei dem Projekt-Hoster *Assembla*. Wenn Sie die Beispiele dieses Buches am Quellcode durcharbeiten wollen, empfehlen wir, so schnell wie möglich die schrittweise Anleitung von Kapitel 5 zur Installation und Inbetriebnahme durchzuführen. Danach können Sie auch ohne Netzzugang, z.B. im Urlaub, daran weiterarbeiten.

¹ Empfohlen sei die Lektüre des Scala-Standardwerkes „Programming in Scala“ von Martin Odersky, Lex Spoon und Bill Venners

1.4. Typographische Hervorhebungen

Im Fließtext werden *wichtige Begriffe* kursiv hervorgehoben. In eine Fließtextzeile eingestreute kurze Codeausschnitte werden durchgängig in Schreibmaschinenschrift oder mit einer durchgängigen Visualisierung von Zwischenräumen dargestellt.

Für die Darstellung der Listings wurde das `listings`-Paket von *LaTeX* verwendet. In der für dieses Buch erstellten Konfiguration generiert es Hervorhebungen für die lexikalischen Einheiten von Scala, HTML, XML und sh. Dabei werden folgende Darstellungen eingesetzt. Im Schwarz/Weiß-Druck werden diese auf entsprechend sinnvolle Grautöne abgebildet.

Einheit	Hervorhebung
Schlüsselwort	Schreibmaschinenschrift blau und fett, z.B. abstract
Bezeichner	Normalschrift schwarz, z.B. <code>noOfLines</code>
String-Literal	Dunkelrot leicht kursiv, Zwischenräume visualisiert, z.B. <i>"ab_cd"</i>
Kommentar	Dunkelgrün voll kursiv, z.B. <i>/*Kommentar*/</i>

Aus technischen Gründen musste darauf verzichtet werden, die einzeichigen Scala-Schlüsselwörter `_` `:` `=` `#` und `@` als solche zu markieren.

2. Scala

Die Programmiersprache Scala, in der das Lift-Framework programmiert ist, ist eine relativ neue funktionale und objektorientierte Sprache. Sie wird seit 2001 unter der Leitung von Martin Odersky entwickelt, der zuvor am Sun-Java-Compiler *javac* und an der Einführung von Generizität in Java arbeitete. Scala-Programme werden in Java-Bytecode übersetzt und sind somit auf Javas virtueller Maschine lauffähig. Dadurch ist Scala kompatibel zu bestehenden Java-Programmen und erlaubt die Benutzung von Java-Frameworks und die Integration von Scala-Komponenten in diese [OdSV08]. Seine Flexibilität bei gleichzeitiger Typstrenge und die Fülle der sich bietenden Möglichkeiten ist jedoch auch mit einem nicht zu unterschätzenden Lernaufwand verbunden. David Pollak, der Erfinder von Lift, äußerte sich dazu in einem Beitrag auf der Lift-Mailingliste: „*It took me 18 months with Scala before I felt comfortable with it and my learning curve with new languages is pretty good (it took 2 weeks to get comfortable with Ruby and 3 months before I felt that I had mastered it.)*” [GoGr09c]

2.1. Eigenschaften

2.1.1. Objektorientierung

Scala ist eine vollständig objektorientierte Sprache. [OdSV08] Dies bedeutet, dass jeder Wert als Objekt angesehen wird. Primitive Datentypen wie in Java existieren dagegen ebenso wenig wie statische Methoden oder Variablen. Stattdessen kann zu jeder Klasse ein gleichnamiges Singleton-Objekt definiert werden, welches man das *Kompagnon*-Objekt der Klasse nennt. In ihm können instanzübergreifende Methoden und Felder unter Einhaltung des objektorientierten Programmierparadigmas definiert werden [Chen09].

Die Struktur von Datentypen wird in Klassen beschrieben, das Konzept der Vererbung ermöglicht wie auch in Java die Erstellung von Klassenhierarchien. Anstelle der Implementierung von Interfaces können Scala-Klassen *Traits* beerben. Diese können neben bloßen Definitionen konkrete Implementierungen von Methoden und Datenfeldern enthalten, sind selbst jedoch nicht instanzierbar. Durch sogenannte *mixin-composition* (dt. etwa „Zusammenset-

zung durch Beimischen“) ist es möglich, auf diese Weise einer Klasse die Eigenschaften beliebig vieler Traits hinzuzufügen.

2.1.2. Funktionale Programmierung

Scala unterstützt funktionale, seiteneffektfreie Programmierung. Funktionen sind in Scala *Werte erster Klasse*.² Sie werden als Objekte betrachtet und haben somit den gleichen Status wie z.B. Variablen. Wie diese können Funktionen innerhalb von Funktionen definiert werden und sowohl als Parameter als auch als Rückgabewert anderer Funktionen dienen. Anonyme Funktionen werden ebenso unterstützt wie echte Closures, die eine „Konservierung“ ihres Definitionskontextes erlauben [Chen09].

2.1.3. Erweiterbarkeit

Der Name Scala steht für „scalable language“ und spielt auf die umfangreichen Erweiterungsmöglichkeiten und die damit verbundene Vielseitigkeit der Sprache an [OdSV08]. Dabei besitzt Scala einen sehr schlanken Sprachkern, bietet aber die Möglichkeit zur Erstellung eigener Bibliotheken, deren Verwendung sich syntaktisch nicht von der eingebauter Sprachkomponenten unterscheidet. Ein wesentlicher Grund dafür ist die Verwendbarkeit eigener Methoden als Infix- und Postfix-Operatoren, für deren Benennung auch viele Sonderzeichen zugelassen sind. So beschreibt der Ausdruck `4 + 5` nichts weiter als den Aufruf der Methode `+` auf den `Int`-Wert `4` mit dem Wert `5` als Parameter. Der Ausdruck ließe sich auch als `(4).+(5)` formulieren. Scala erlaubt nämlich den Verzicht auf den Punkt vor einem Methodenaufruf sowie die Klammern um dessen Parameter, solange es sich nur um *einen* Parameter handelt und der Empfänger des Methodenaufrufs explizit angegeben ist. Daher ist `println "hallo"` nicht zulässig, während `Console.println "hallo"` ausgeführt werden kann und den selben Effekt wie `println("hallo")` (bzw. `Console.println("hallo")`) erzielt [OdSV08].

2.1.4. Statisches Typsystem

Scala ist statisch typisiert und kann somit die Vorteile statischer Typ-Systeme nutzen, zu denen die frühe Fehlererkennung zur Kompilierzeit, die verbesserte Unterstützung für sichere Refaktorisierung und die Anfertigung von Programmdokumentationen gehören [OdSV08]. Durch die ausgeprägte Fähigkeit der Typinferenz wird in Scala der häufig kritisierte erhöhte

² Zu dem Begriff siehe http://en.wikipedia.org/wiki/First-class_object

Schreibaufwand statischer Typisierung auf ein Minimum reduziert. So erkennt der Scala-Compiler bei der Initialisierung von Variablen den Typ des zugewiesenen Wertes und typisiert die Variable entsprechend. `var i = 42` wird demzufolge als `var i : Int = 42` interpretiert. Bei der Typisierung von Funktionen funktioniert dieser Mechanismus ebenso, mit der Ausnahme rekursiver Funktionen. Sie erfordern eine explizite Angabe des Rückgabewert-Typs.

Wie Java unterstützt Scala generische Klassen.

2.1.5. XML-Unterstützung

Scala unterstützt die Verwendung und Manipulation von XML innerhalb des Programmcodes. Wohlgeformtes XML kann überall dort formuliert werden, wo ein Ausdruck erlaubt ist. Das Resultat eines XML-Ausdrucks ist vom Typ `Elem` bzw. `NodeSeq`, einer Sequenz aus XML-Elementen. Lift macht ausgiebig Gebrauch von Scalas XML-Unterstützung. In einen XML-Ausdruck können Scala-Berechnungen eingestreut werden, indem diese in geschweifte Klammern eingeschlossen werden:

```
//Datei src/test/scala/platform/PlatformTest.scala, Ausschnitt xmlInsert:  
val myXml: scala.xml.NodeSeq = <span>{gruss + " " + name}</span>
```

Listing 2.1: Scala-Berechnung in einem XML-Ausdruck

2.2. Entwicklungsumgebungen

Es existieren Scala-Plugins für die Entwicklungsumgebungen *Eclipse*, *NetBeans* und *IntelliJ IDEA* sowie für diverse Editoren wie *Emacs* oder *vim* [ChDW09]. Obwohl es im letzten Jahr deutliche Fortschritte gab, reichen die Scala-Plugins noch nicht an die entsprechenden Java-IDEs heran, was Konfigurationseinfachheit, Stabilität, Komfort und Funktionsumfang betrifft. Vor der Installation eines Scala-Plugins in eine IDE sollte man sich auf der Seite des Herstellers über die genau zueinander passenden Versionen der IDE und des Plugins informieren.

Die Beispielanwendung wurde ursprünglich mit Eclipse (Version 3.5.1) und dessen Scala-Plugin (Version 2.7.7.final) erzeugt. In dieser Version kam es häufig zu Unregelmäßigkeiten bei der Darstellung des Programmcodes. Die Syntax-Hervorhebung funktionierte oft nicht fehlerfrei, XML-Elemente mussten mit Klammern umgeben werden, um als solche erkannt zu werden. Gelegentlich war ein erneutes Öffnen einer bearbeiteten Datei nötig, um fälschlich angezeigte Fehler zu beseitigen. Besonders beim Erlernen von Scala und Lift stellte

dieser Umstand ein zusätzliches Hindernis dar, da entweder valider Code als fehlerhaft gekennzeichnet wurde, oder Fehler nicht als solche kenntlich gemacht wurden.

Ab Scala 2.8 stellt der Scala-Compiler ein umfangreiches API (*presentation compiler*) für die Nutzung durch IDEs bereit, was zu einer Vereinheitlichung der Kompilation innerhalb und außerhalb der IDE führen soll [Scal10].

2.3. Leistung

Die Ausführungsgeschwindigkeit von Scala-Programmen wird in dem Buch *Programming in Scala*[OdSV08] als „im Allgemeinen gleich auf mit [der von] Java-Programmen“ beschrieben. Das *Computer Language Benchmarks Game* bestätigt diese Aussage in der Tendenz bei höherem Speicheraufwand und geringerem Codeumfang [CLBG10].

3. Lift

Bevor in den nächsten Kapiteln konkret auf die Entwicklung von Webanwendungen mit Lift eingegangen wird, soll an dieser Stelle ein kurzer Überblick über die Eigenschaften des Frameworks gegeben werden, mit dessen Entwicklung David Pollak im Jahr 2007 begann.

3.1. Vorbilder

Lift erhebt den Anspruch, die fortschrittlichsten Konzepte moderner Webframeworks zu vereinen. Es folgt eine Auflistung von Webframeworks, deren Merkmale als Vorbilder bei der Entwicklung von Lift gedient haben [Gavi08]:

- **Ruby on Rails:** Die Grundprinzipien *Convention over Configuration* und *DRY: Don't repeat yourself*, nach denen sich die Konfiguration einer Anwendung durch die Einhaltung von Konventionen (z.B. bei der Verwendung von Verzeichnissen für bestimmte Programmteile oder der Benennung von Dateien) ergibt, und jede Wiederholung einmal geschriebenen Programmcodes zu vermeiden ist, bilden die Grundlage für die agile Softwareentwicklung und werden in Ruby on Rails konsequent verfolgt. [RuTH08] In Lift spiegeln sich diese Ansätze an unterschiedlichen Stellen wieder, z.B. bei der Benennung internationalisierter Templates, der Verwendung von Maven für die Projektverwaltung, der Paketstruktur und der Auffindung von Klassen durch XHTML-Tags.
- **Seaside:** Das in der Programmiersprache Smalltalk implementierte Webframework Seaside ermöglicht die Entwicklung zustandbehalteter Web-Applikationen (im Gegensatz zu dem Ansatz, Zustände nach jedem Request in einer Datenbank zwischenzuspeichern) [Seas]. Der Einsatz von Closures und Callback-Funktionen in Lift wurde durch Seaside inspiriert und bildet die Basis für die Unterstützung der AJAX- und Comet-Funktionalität in Lift sowie für die Erstellung und Verarbeitung von Formularen.
- **Wicket:** Die ausschließliche Verwendung valider XHTML-Elemente innerhalb von Templates ist ein Merkmal des Wicket-Frameworks welches in Lift übernommen wurde. Templates sollen durch den Wegfall unbekannter Elemente leichter von Designern

bearbeitet werden könne. Zudem soll die Bearbeitung der Templates mit standardisierten HTML-Editoren ermöglicht werden [Apac09a].

- **Django:** Das Python-basierte Webframework Django bietet die Möglichkeit automatisch einen umfangreichen Administrationsbereich erstellen zu lassen. Dieser beinhaltet eine Benutzerverwaltung sowie zahlreiche Ansichten für die Bearbeitung und Erzeugung von Datensätzen [Djan10]. Die integrierte Benutzerverwaltung und die Möglichkeit zur Erstellung von CRUD-Seiten in Lift haben diese Funktionalität zum Vorbild.

3.2. Nutzung von Scala-Features

Die Ausprägung von Lift ist maßgeblich von Scala beeinflusst:

- Lift-Anwendungen werden als Web-Archive (.war) exportiert und können dank der Kompatibilität von Scala zu Java auf beliebigen Servlet-Containern installiert werden.
- Bei der Entwicklung kann auf eine breite Auswahl an vorhandenen Java-Bibliotheken zurückgegriffen werden.
- Die Typ-Inferenz von Scala ermöglicht es in kurzer Zeit ausdrucksstarken und typischeren Programmcode zu erzeugen.
- Durch die integrierte XML-Unterstützung kann XHTML-Code komfortabel und typischer innerhalb von Scala-Programmen manipuliert und erstellt werden.
- Lift macht häufigen Gebrauch von Closures und deren Eigenschaft, ihren Definitionskontext „konservieren“ zu können.
- Die Comet-Unterstützung basiert wesentlich auf dem *Actor*-Konzept von Scala.
- Die Java-Kompatibilität ermöglicht die Projektverwaltung durch das Build-Management-Tool Maven.

4. Maven und Lift

Für die Verwaltung von Lift-Projekten wird das Build-Management-Tool *Maven* eingesetzt. Maven ist ein mächtiges Werkzeug, auf dessen Funktionsweise hier aufgrund seines Umfangs nur oberflächlich eingegangen werden kann. Es sollen vor allem die Aspekte beleuchtet werden, die bei der Entwicklung von Lift-Applikationen von Bedeutung sind.

4.1. Kurzbeschreibung

Maven bietet die Möglichkeit sogenannte *Goals* zu definieren, mit denen häufig durchgeführte Arbeitsschritte automatisiert durchgeführt werden können. Folgende Arbeitsschritte kommen bei der Entwicklung von Lift-Projekten zum Einsatz:

- Projekterstellung anhand von Vorlagen
- Projektanpassung für die Bearbeitung in Entwicklungsumgebungen
- Kompilierung
- Erstellung von Dokumentationen
- Deployment und Inbetriebnahme

Abhängigkeiten von anderen Software-Bibliotheken werden durch die Nutzung von Repositories aufgelöst. Beim Build-Prozesses wird dadurch die Verwendung aktueller Versionen gewährleistet [Apac09].

4.2. Konfiguration

Die Konfiguration eines Maven-Projekts erfolgt über das *Project Object Model* (POM) in einer Datei `pom.xml` im Projekt-Hauptverzeichnis. In ihr sind alle Informationen, die für die Verwaltung des Projekts mit Maven benötigt werden, enthalten. Die projektübergreifende Konfiguration von Maven erfolgt in der Datei `.m2/settings.xml` im HOME-Verzeichnis des Benutzers. Darin sollte für Deutschland insbesondere ein nahe gelegener Spiegelserver des Maven Central Repository angegeben werden:

```
<settings>
  <mirrors>
    <!-- Einstellung eines Mirrors des Maven Central Repository in Deutschland: -->
    <mirror>
      <id>netcologne.de</id>
      <url>http://mirror.netcologne.de/maven2</url>
      <mirrorOf>central</mirrorOf>
      <!-- Germany, Cologne -->
    </mirror>
  </mirrors>
</settings>
```

Listing 4.1: Projektübergreifende Maven-Einstellungen

4.3. Repositories

Maven nutzt ein lokales Repository, welches als Zwischenspeicher für Dateien aus entfernten Repositories und als Austauschverzeichnis für die Ergebnisse (*Artefakte*) der Goals dient. Es wird durch Maven standardmäßig in `.m2/repository` im HOME-Verzeichnis des Benutzers angelegt. Innerhalb der POM-Datei können entfernte Repositories definiert werden, in denen nach Plugins und Software-Bibliotheken gesucht werden soll.

4.4. Plugins

Der Funktionsumfang von Maven kann durch den Einsatz von Plugins erweitert werden, die ebenfalls dynamisch aus Repositories eingebunden werden. Für die Arbeit mit Lift spielen die folgenden Plugins eine wichtige Rolle:

Scala-Plugin Zur Inbetriebnahme einer Lift-Anwendung mit Maven ist keine manuelle Installation von Scala erforderlich. Die in der POM-Datei angegebene Scala-Version wird dank des Scala-Plugins automatisch aus dem Repository geladen. Das Plugin bietet zusätzliche Funktionen, die über Maven gesteuert werden können, zum Beispiel die Erstellung einer Programmdokumentation.

Jetty-Plugin Der Servlet-Container *Jetty* ist aufgrund seiner leichtgewichtigen Architektur sehr gut für die Integration in Software geeignet und wird aus diesem Grund als Default-

Server für Lift-Anwendungen verwendet. Durch die voll automatisierte Konfiguration wird der Entwicklungsprozess erleichtert.

Derby-Plugin Als voreingestellte Datenbank wird in Lift die Datenbank *Derby* genutzt. Derby bietet ähnliche Vorteile wie der Jetty-Server: eine leichtgewichtige Architektur und ein minimaler Konfigurationsaufwand.

Eclipse-Plugin Das Eclipse-Plugin bietet die Möglichkeit, ein vorhandenes Lift-Projekt für den Import in die Eclipse-Entwicklungsumgebung vorzubereiten.

4.5. Die POM-Datei

Eine POM-Datei besitzt folgende Grundstruktur:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion><!-- Version des POM-Objekt-Modells --></modelVersion>
  <groupId>
    <!-- Weltweit eindeutiger Bezeichner des Herstellers: meistens inverser Domänenname -->
  </groupId>
  <artifactId><!-- Name des Hauptartefakts (des WAR-Archivs)</artifactId>
  <name><!-- Name des Projekts --></name>
  <version><!-- Projektversion --></version>
  <packaging><!-- Ausgabeformat des Artefakts --></packaging>
  <build>
    <plugins>
      <!-- Build-Plugins -->
    </plugins>
  </build>
  <properties>
    <!-- Eigenschaften des Projekts -->
  </properties>
  <reporting>
    <plugins>
      <!-- Plugins für die Erstellung von Dokumentationen -->
    </plugins>
  </reporting>
  <dependencies>
    <!-- Definition der Abhängigkeiten -->
  </dependencies>
</project>
```

Listing 4.2: Grundstruktur einer POM-Datei

Die POM-Datei für Lift-Projekte besitzt weitere Elemente. Werden Änderungen an der POM-Datei benötigt, z.B. für die Nutzung bestimmter Web-Server oder Datenbanken, wird im entsprechenden Abschnitt ausdrücklich darauf eingegangen.

4.6. Einsatz in Lift

Maven stellt für den gesamten Lebenszyklus eines Lift-Projekts *Goals* zur Verfügung, die an dieser Stelle vorgestellt werden.

4.6.1. Erstellung eines neuen Lift-Projekts

Um ein neues Lift-Projekt zu erstellen, stehen sogenannte *Archetypen* zur Verfügung. Archetypen sind Vorlagen für die Erstellung von Maven-Projekten. Sie geben eine Verzeichnissstruktur und benötigte Einstellungen für das Projekt vor. Für die Erstellung neuer Lift-Projekte können die Archetypen „lift-blank“ und „lift-basic“ genutzt werden. Bei „lift-blank“ handelt es sich um ein leeres Projekt, „lift-basic“ beinhaltet bereits eine Benutzerverwaltung und CSS-Vorlagen für die grafische Gestaltung der Anwendung.

Durch die Eingabe des folgenden Befehls auf einer Linux-Kommandozeile wird ein Lift-Projekt aus dem Archetype „lift-basic“ erstellt. (Unter Windows ist für mehrzeilige Kommandos statt \ der Circonflex ^ zu verwenden.)

```
# Datei demo/liftBasic.sh, Ausschnitt command:
mvn archetype:generate \
  -DarchetypeGroupId=net.liftweb \
  -DarchetypeArtifactId=lift-archetype-basic_2.8.0 \
  -DarchetypeVersion=2.1 \
  -DarchetypeRepository=http://scala-tools.org/repo-releases \
  -DremoteRepositories=http://scala-tools.org/repo-releases \
  -DgroupId=com.company \
  -DartifactId=myproject \
  -Dversion=0.1-SNAPSHOT
```

Listing 4.3: Befehl zur Erstellung eines Lift-Projekts mit dem lift-basic Archetyp

Die Projekteigenschaften werden angezeigt und müssen mit y (yes) bestätigt werden. Es wird eine Verzeichnishierarchie wie in Listing 4.4 erstellt.

Der Name des Projektverzeichnisses wird durch den Parameter „artifactId“ bestimmt. Innerhalb des Ordners `myproject/src/main/scala` wird die Paketstruktur `com.company`, die durch den „groupId“-Parameter festgelegt wird, in entsprechenden Verzeichnissen abgebildet.

4.6.2. Kompilierung

Um ein Lift-Projekt zu kompilieren, steht das Kommando `mvn compile` zur Verfügung. Die kompilierten Klassen werden im Verzeichnis `target/classes` abgelegt.

4.6.3. Deployment und Inbetriebnahme

Die Anwendung kann mit dem Befehl `mvn jetty:run` gestartet werden. Dieses Kommando schließt Kompilierung und Test mit ein. Lift-Anwendungen laufen voreingestellt auf einem Jetty-Server und benutzen eine H2-Datenbank.^{3 4} Die dafür benötigten Bibliotheken sind in der POM-Datei als Abhängigkeiten definiert und werden aus dem Repository automatisch geladen.



Abbildung 4.1.: Startseite des generierten Lift-Projekts mit dem „lift-basic“-Archetyp

Lift-Anwendungen sind auf jedem Java-Servlet-Container lauffähig [Liftb]. Um lediglich ein WAR-Archiv der Anwendung zu packen, kann das Maven-Kommando `mvn package` ausgeführt werden. Das Archiv wird im Ordner `target` abgelegt und kann anschließend in jedem

³ Die Datenbank-Konfiguration von Lift-Anwendungen wird im Kapitel 11 beschrieben.

⁴ Die Beispielanwendung nutzt ebenfalls eine H2-Datenbank.

```

myproject
+-- pom.xml
+-- project
|  +-- build
|  |  +-- LiftProject.scala
|  +-- build.properties
+-- src
    +-- main
        |  +-- resources
        |  +-- scala
        |  |  |-- bootstrap
        |  |  |  +-- liftweb
        |  |  |  |  +-- Boot.scala
        |  |  +-- com
        |  |  |  +-- company
        |  |  |  |  +-- hellolift
        |  |  |  |  |  +-- comet
        |  |  |  |  |  +-- lib
        |  |  |  |  |  +-- model
        |  |  |  |  |  |  +-- User.scala
        |  |  |  |  |  +-- snippet
        |  |  |  |  |  |  +-- HelloWorld.scala
        |  |  |  |  +-- view
        |  +-- webapp
        |  |-- WEB-INF
        |  |  +-- web.xml
        |  +-- index.html
        |  +-- templates-hidden
        |  +-- default.html
    +-- test
        +-- resources
        +-- scala
            +-- LiftConsole.scala
            +-- RunWebApp.scala
            +-- org
                +-- hellolift
                    +-- AppTest.scala

```

Listing 4.4: Verzeichnisstruktur des erstellten Projekts

gewünschten Java-Servlet-Container in Betrieb genommen werden. Im Rahmen der Erstellung der Arbeit wurde der Einsatz der Beispielanwendung auf einem Tomcat-Server (Version 5.5.28) getestet. Die Inbetriebnahme mit dessen „Tomcat Web Application Manager“ verlief problemlos.

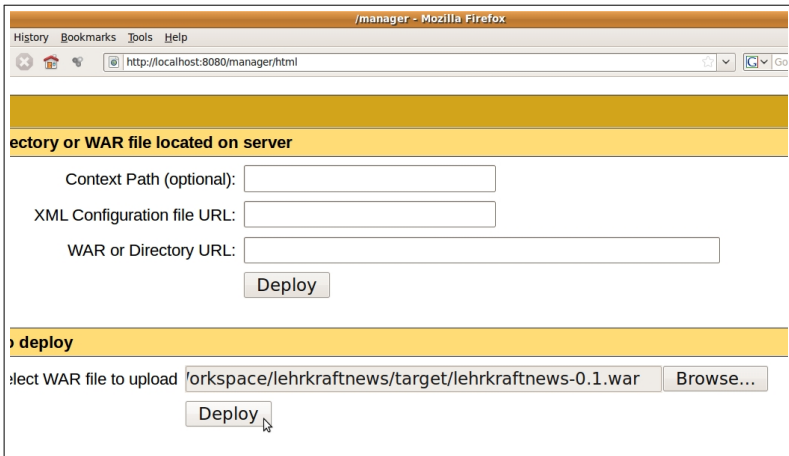


Abbildung 4.2.: Inbetriebnahme der Beispielanwendung über Tomcats „Web Application Manager“

4.6.4. Bearbeitung des Projekts für die Nutzung in einer IDE

Für die Bearbeitung eines Maven-Projekts in einer IDE wie *IntelliJ IDEA* oder *Eclipse* stehen grundsätzlich zwei Wege zur Verfügung.

- IDE-Projekt generieren mittels Maven-Plugin, z.B. durch die Kommandos `mvn idea:idea` oder `mvn eclipse:eclipse`.
- Maven-Projekt importieren mittels IDE-Plugin.

Wir haben wegen höheren Komforts den letzten Weg bevorzugt. In *IntelliJ IDEA* ist ein Maven-Plugin schon vorinstalliert, in *Eclipse* benutzt man dazu das Plugin *m2eclipse*. Näheres siehe im Abschnitt A.1. Abbildung 4.3 zeigt die Darstellung der Projektstruktur im Eclipse-Package-Explorer.

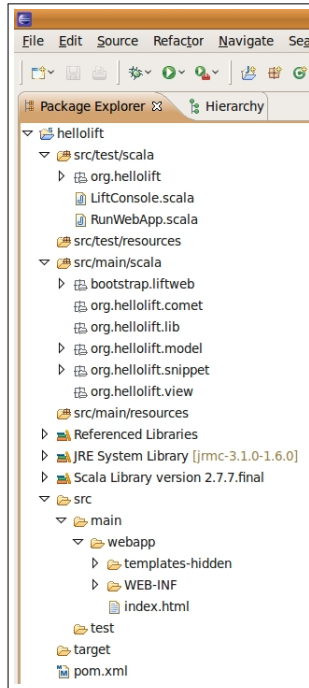


Abbildung 4.3.: Darstellung der Projektstruktur in Eclipse

4.6.5. Erstellung einer Quelltext-Dokumentation

Um eine HTML-Dokumentation für den Programm-Code des Lift-Projekts zu erstellen, kann der Befehl `mvn scala:doc` genutzt werden.

Diese Funktionalität wird von Scala und dem Maven-Scala-Plugin [ScTo09] bereitgestellt. Die Dokumentation wird in dem Ordner `target/site/scaladocs` abgelegt.

5. Die Beispielanwendung „Lehrkraftnews“

Bei der Implementierung der Beispielanwendung „Lehrkraftnews“ wurde darauf Wert gelegt, den größtmöglichen Teil der Möglichkeiten, die Lift bei der Erstellung von Web-Applikationen bietet, abzudecken. Ein wesentliches Merkmal des Lift-Frameworks ist die hervorragende Unterstützung für den asynchronen Datenaustausch durch AJAX- und Comet-Funktionalitäten [ChDW09]. Eine breitere und einseitigere Nutzung dieser Technologien innerhalb der Beispielanwendung wäre möglich, widerspricht aber dem Vorsatz, eine lauffähige Anwendung bereitzustellen, anhand derer alle wesentlichen Aspekte der Entwicklung mit dem Lift-Framework erklärt werden. Die Beispielanwendung ist an die Anwendung „Lehrkraftnews“ des Fachbereichs VI der *Beuth-Hochschule für Technik Berlin* angelehnt.⁵

5.1. Kurzbeschreibung

Bei der Beispielanwendung handelt es sich um ein Benachrichtigungssystem für Lehrkräfte und Studenten. Lehrkräfte können Nachrichten erstellen und diese per E-Mail an Abonnenten versenden. Benutzer können die Nachrichten in verschiedenen Ansichten betrachten und Abonnements verwalten.

5.1.1. Benutzerverwaltung

Der Anwendung liegt eine Benutzerverwaltung zugrunde. Benutzer werden anhand ihrer E-Mail-Adresse und eines Passworts identifiziert. Benutzer besitzen die Möglichkeit, ihr Profil zu ändern. Administratoren können Benutzerprofile bearbeiten und löschen.

⁵ Für diese existiert eine von Prof. Christoph Knabe angefertigte Fallstudie. Trotz der Abweichungen der Beispielanwendung dieses Buches in einigen Punkten bietet die Fallstudie einen zusätzlichen Überblick über die zugrunde liegende Idee der Anwendung:
<http://public.beuth-hochschule.de/~knabe/fach/swp-i/lehrkraftnews/>

Registrierung

Benutzer können sich selbstständig mit ihrem Namen und ihrer E-Mail-Adresse registrieren. Die E-Mail-Adresse wird durch eine Bestätigungs-E-Mail auf ihre Echtheit geprüft. Neu registrierten Benutzern wird die Rolle „Student“ zugewiesen. Administratoren können Benutzern neue Rollen zuweisen.

Benutzerrollen

Es gibt drei mögliche Rollen, die ein registrierter Benutzer besitzen kann: „Student“, „Lehrkraft“ oder „Administrator“. Jeder Benutzer durchläuft den gleichen Registrierungsprozess. Ein Wechsel der Rolle kann durch Benutzer mit der Rolle „Administrator“ vorgenommen werden. Ein solcher Benutzer wird vor Inbetriebnahme der Anwendung automatisiert erstellt (siehe Abschnitt 5.2.3).

Abbildung 5.1 bietet eine Übersicht über die Anwendungsfälle für die verschiedenen Benutzerrollen.

5.1.2. Nachrichten

Nachrichten können nur durch Lehrkräfte erstellt werden. Sie besitzen neben einem Verweis auf ihren Autor einen Nachrichtentext und ein Gültigkeitsdatum. Einmal erstellt, können sie nur von ihrem Autor und von Administratoren bearbeitet oder gelöscht werden.

5.1.3. Abonnements

Lehrkräfte können abonniert werden, Abonnenten können Studenten oder andere Lehrkräfte sein. Abonnements stellen eine Verbindung zwischen einem Benutzer (Student oder Lehrkraft) und einer Lehrkraft dar. Vorhandene Abonnements können jederzeit gekündigt werden.

5.1.4. Zugrunde liegendes Datenmodell

Die Abbildungen 5.2 und 5.3 verdeutlichen das der Anwendung zugrunde liegende Datenmodell und dessen Umsetzung in der Datenbank.

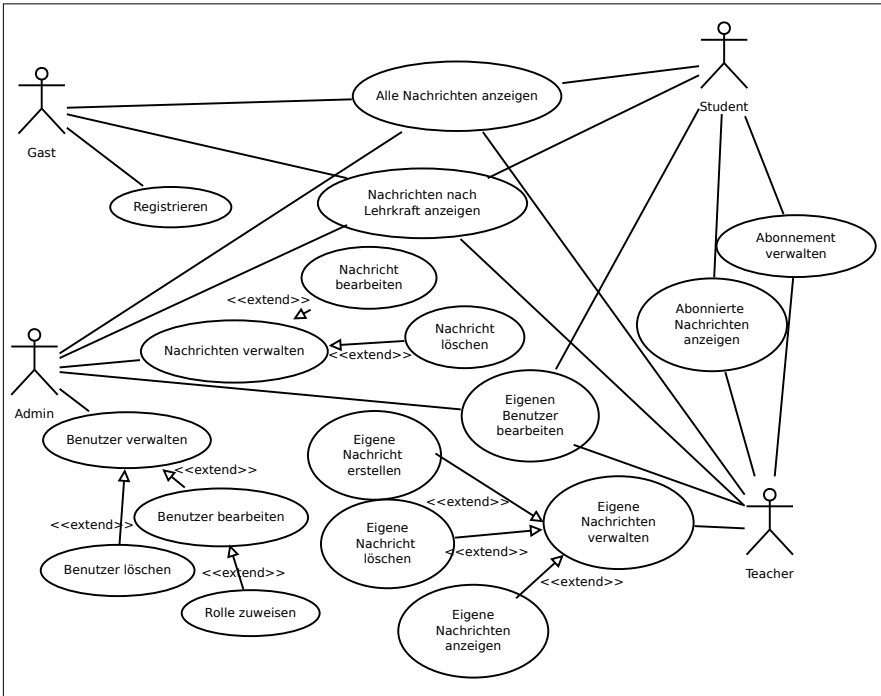


Abbildung 5.1.: Rollenbasiertes Anwendungsfalldiagramm für die Beispielanwendung

5.1.5. Besonderheiten

Ausgabe asynchroner Statusmeldungen beim E-Mail-Versand

Als Demonstration für die Comet-Technologie und deren Nutzbarkeit in Lift dient der E-Mail-Versand von Nachrichten an Abonnenten. Der Autor einer Nachricht erhält während des nebenläufigen E-Mail-Versandes fortlaufend Rückmeldungen über die Anzahl der noch zu versendenden E-Mails und über die Empfängeradressen der erfolgreich versandten E-Mails. Beim Auftreten von `SendFailedExceptions`, geworfen durch die Methode `send` der für den E-Mail-Versand genutzten Java-Klasse `javax.mail.Transport`, werden zusätzlich fortlaufend Rückmeldungen mit den fehlerhaften Empfängeradressen angezeigt.

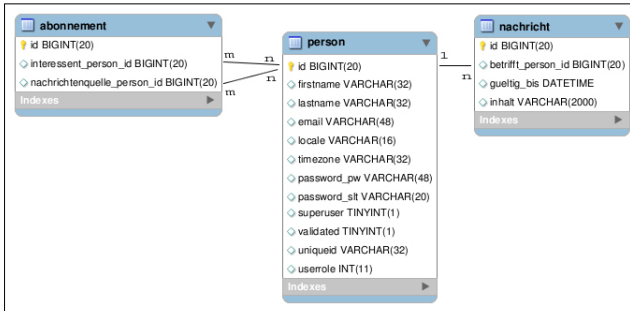


Abbildung 5.2.: MySQL - Datenbankschema der Beispielanwendung

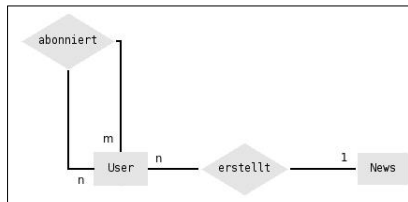


Abbildung 5.3.: Entity/Relationship-Modell für die Beispielanwendung

Benachrichtigung über neue Nachrichten

Benutzer der Anwendung werden unter Verwendung der Comet-Technologie unmittelbar nach der Erstellung einer neuen Nachricht über deren Vorhandensein informiert. Eine aktive Abfrage ist dafür nicht nötig.

5.2. Installation und Inbetriebnahme

Der Installationsverlauf variiert plattformabhängig. Im folgenden wird die Installation auf einem Debian-Linux-System beschrieben. Die Kommandos zur Installation werden von der Kommandozeile ausgeführt.

Voraussetzung für die Inbetriebnahme der Beispielanwendung ist die Installation nachfolgender Software:

- Java Development Kit (JDK) (Version 6 und aufwärts)
- Maven 2

Außerdem wird ein Internetzugang benötigt, da Maven umfangreiche zusätzliche, für die Lauffähigkeit des Projekts benötigte, Software aus entfernten Repositories herunterladen muss.

5.2.1. Installation der benötigten Software

Java SE Development Kit (JDK)

Das JDK bietet Kommandozeilenwerkzeuge für die Entwicklung von Java-Applikationen. Es kann von Oracle bezogen werden.⁶ Es sollte das JDK 6 oder später gewählt werden. Zunächst muss die Installationsdatei ausführbar gemacht werden.⁷ Auch hier gehen wir von einem Linux-System aus:

```
$ chmod 755 jdk-6u17-linux-x64.bin
```

Die anschließende Ausführung der Datei entpackt die Software in das Verzeichnis `jdk1.6.0_17`:

```
$ ./jdk-6u17-linux-x64.bin
```

Auf dem Linux-System Ubuntu 10 gibt es schon ein vorgefertigtes Paket `sun-java6-jdk`, welches einfacher als oben beschrieben, mittels *Synaptic* installiert werden kann.

Die Anpassung der Umgebungsvariablen `JAVA_HOME` und `PATH` schließt die Installation ab. Für das Setzen der Umgebungsvariablen über die Dauer der Ausführung des Terminals hinaus empfiehlt sich die Unterbringung der `export`-Befehle in einem Skript.⁸

```
$ export JAVA_HOME=/[...]/jdk1.6.0_17
$ export PATH=$JAVA_HOME/bin:$PATH
```

Durch die Ausgabe der aktuellen Java-Version in einem neu geöffneten Shell kann der Erfolg der Installation bestätigt werden.

```
$ java -version
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
Java HotSpot(TM) 64-Bit Server VM (build 14.3-b01, mixed mode)
```

6 <http://www.oracle.com/technetwork/java/javase/>

7 Dies erfordert das Kopieren der Datei auf die Festplatte, z.B. das HOME-Verzeichnis

8 Bei der Verwendung der `bash`-Shell bietet sich dafür die Datei `$HOME/.bashrc` an.

Maven

Maven ist ein kommandozeilenorientiertes Build-Werkzeug, mit dem aus Quelldateien ein auslieferungsfähiges Produkt erstellt werden kann. Es kann von Apache bezogen werden⁹. Kopieren Sie das Verzeichnis `apache-maven-2.2.1` auf die Festplatte. Auf dem Debian-Linux-Derivat Ubuntu 10 gibt es auch ein vorgefertigtes Paket `maven2`, welches mittels *Synaptic* installiert werden kann. Nun müssen noch die Umgebungsvariablen `M2_HOME`, `M2` und `PATH` gesetzt bzw. angepasst werden. Das lokale Maven-Repository wird per Default-Einstellung im `HOME`-Verzeichnis des Benutzers angelegt (`~/.m2`)¹⁰

```
$ export M2_HOME=[...]/apache-maven-2.2.1
$ export M2=$M2_HOME/bin
$ export PATH=$M2:$PATH
```

Anschließend kann geprüft werden, ob die Installation erfolgreich war:

```
$ mvn -v
Apache Maven 2.2.1 (r801777; 2009-08-06 21:16:01+0200)
Java version: 1.6.0_17
Java home: /home/tom/Desktop/jdk1.6.0_17/jre
```

git

*git*¹¹ ist ein Kommandozeilenwerkzeug zur verteilten Versionsverwaltung. Im Gegensatz zu Subversion oder CVS muss es kein zentrales Repository mit allen Versionen des Projekts geben. Vielmehr ist jede Arbeitskopie ein vollgültiges Repository mit der gesamten Versionsgeschichte und man kann sich Änderungen von jedem beliebigen Repository holen (`git pull`) bzw. dorthin schicken (`git push`).

git sollte bei Linux-Systemen schon vorhanden sein bzw. kann mit dem zugehörigen Paket installiert werden. Auf Ubuntu 10 kann man es als Paket `git-core` mittels *Synaptic* installieren. Für die Installation unter Windows sind die Hinweise des Hosters *Assembla*¹² sehr instruktiv.

⁹ <http://maven.apache.org/>

¹⁰ Das Repository wird erst angelegt, wenn Maven zum ersten Mal Daten herunterlädt.

¹¹ <http://git-scm.com/>

¹² <https://www.assembla.com/code/liftbuchcode/git/repo/instructions>

5.2.2. Inbetriebnahme der Anwendung

Der komplette, funktionsfähige Quellcode der Beispielanwendung „Lehrkraftnews“ ist bei dem Projekthoster *Assembla* in einem git-Repository¹³ abgelegt. Die aktuelle Revision kann in einer Web-Darstellung betrachtet werden.¹⁴ Folgen Sie den dort angegebenen *Instructions* und klonen Sie das git-Repository mittels Kommando

```
git clone git://git.assembla.com/liftbuchcode.git
```

Dies legt ein Unterverzeichnis `liftbuchcode` an, in dem sich das Maven steuernde Projekt-Objekt-Modell `pom.xml` und der Quelltextordner `src` befinden. Dieses ist zunächst auf dem aktuellsten Stand, d.h. wahrscheinlich aktueller als die Ausgabe dieses Buches, die Ihnen gerade vorliegt. Sie können die Historie der in Ihrer Arbeitskopie wirksamen Commits mittels des Kommandos `git log` sehen. Zuerst erscheint das neuste wirksame Commit, danach die älteren.

Um exakt mit der Softwareversion zu arbeiten, auf die der Text des Ihnen vorliegenden Buches abgestimmt ist, schalten Sie bitte mittels `git checkout` auf die entsprechende *Commit-Id*.

```
git checkout c7fbe0675343a682e291dc2c4aea7921e2463308
```

Listing 5.1: Umschalten der git-Arbeitskopie auf die zu dieser Buchversion passende *Commit-Id*

Dabei dürfen Sie auch die angegebene hexadezimale Commit-Id abkürzen, sofern der Anfang innerhalb aller Commit-Ids eindeutig ist, hier also z.B. auf 5 Hex-Ziffern:

```
git checkout c7fbe
```

Listing 5.2: Umschalten der git-Arbeitskopie mit Hilfe der verkürzten *Commit-Id*

Wenn Sie aber wieder mit der neusten Software-Version arbeiten wollen, geben Sie ein:

```
git checkout master
```

¹³ [git://git.assembla.com/liftbuchcode.git](https://git.assembla.com/liftbuchcode.git)

¹⁴ <https://www.assembla.com/code/liftbuchcode/git/nodes?rev=master>

Mail-Konfiguration

Der Versand von E-Mails mit der Beispielanwendung setzt voraus, dass die Zugangsdaten für den verwendeten SMTP-Server korrekt eingestellt sind. Damit diese Authentifizierungsdaten nicht in einem öffentlichen Quelltext-Repository stehen, liest die Applikation diese und die anderen, für *JavaMail*¹⁵ nötigen Properties aus der Datei `mail.properties` im Home-Verzeichnis des Benutzers. Diese Datei sieht mit Ausnahme der mit `my` beginnenden Teile bei Benutzung des Mail-Servers der Beuth-Hochschule Berlin¹⁶ wie folgt aus:

```
mail.smtp.host=mail.beuth-hochschule.de
mail.smtp.starttls.enable=true
mail.smtp.auth=true
mail.username=myUsername
mail.password=myPassword
#Mail-Adressen von 3 Beispielbenutzern, die in der Testsuite erzeugt werden:
mail.teacher.1=myName1@myDomain.de
mail.teacher.2=myName2@myDomain.de
mail.student=myName3@myDomain.de
```

Listing 5.3: Mail-Zugangskonfiguration in `mail.properties`

Die mit `mail.smtp.` beginnenden Properties sind in *JavaMail* standardisiert. Die Property `mail.smtp.host` gibt die Adresse des SMTP-Servers an. Die beiden folgenden Properties sind wahrscheinlich auch bei anderen Mailservern unverändert sinnvoll. Die Properties `mail.username` und `mail.password` sind Lehrkraftnews-spezifisch. Sie geben Benutzernamen und Passwort des für das Versenden von Nachrichten zu benutzenden E-Mail-Kontos an. Die mit `mail.teacher` oder `mail.student` beginnenden Properties werden durch die Testsuite (siehe folgenden Abschnitt) als Benutzer-Identifikationen für einige Beispielbenutzer verwendet.

Erstellen Sie eine Datei `HOME/mail.properties` mit den in Listing 5.3 abgedruckten Zeilen und passen Sie alle Property-Werte für den konkret zu benutzenden Mailserver an! Geben Sie bei `mail.teacher.1`, `mail.teacher.2` und `mail.student` drei E-Mail-Adressen an, auf die Sie zugreifen können, um den tatsächlichen Mailversand überprüfen zu können.

Die Datei `HOME/mail.properties` wird in der `boot`-Methode der Klasse `Boot` (Datei `src/main/scala/bootstrap/liftweb/Boot.scala`) ausgewertet. Dies wird im Abschnitt 11.2 beschrieben.

¹⁵ <http://www.oracle.com/technetwork/java/index-jsp-139225.html>

¹⁶ <http://www.beuth-hochschule.de/>

Testlauf und Füllen der Datenbank mit Beispieldaten

Die Beispielanwendung nutzt eine H2-Datenbank¹⁷, die in die Anwendung eingebettet ist. Somit entfällt die Installation und Konfiguration eines SQL-Servers und die Anwendung ist sofort lauffähig.

Wenn Sie nicht mit einer leeren Datenbank beginnen möchten, ist es sinnvoll, zuerst die Testsuite durchzuführen. Die darin befindlichen Testfälle `UserTest` und `NewsTest` löschen den gesamten Datenbankinhalt und tragen den Administrator `admin@admin.de` mit dem Passwort `admin` sowie 5 Lehrkräfte und einen Studenten (alle mit dem Passwort `passwort`) und einige Nachrichten ein. Dies geschieht wie abgedruckt in Listing 5.4. Bitte vergessen Sie nicht die im vorigen Abschnitt erwähnte Definition der drei Beispielbenutzeradressen, damit Sie den Erfolg des Test-Versandes überprüfen können. Die angegebenen Mail-Adressen der Politiker sind Phantasieadressen, um die Fehlerbehandlung beim Mailversand testen zu können (Siehe Comet-Kapitel 10.2).

```
//Datei src/test/scala/com/lehrkraftnews/model/DatabaseUtil.scala, Ausschnitt createUsers:  
boot.makeSureAdminExists  
//val mailPropertyEntries = MailProperties.getAll  
_createUser(User.Teacher, "Knabe", "Christoph", MailProperties("mail.teacher.1"))  
_createUser(User.Teacher, "Triel", "Bernd_Holger", MailProperties("mail.teacher.2"))  
_createUser(User.Student, "Heinrich", "Theodor_Friedrich", MailProperties("mail.student"))  
_createUser(User.Teacher, "Merkel", "Angela", "Angela.Merkel@bundesdeutschland.de")  
_createUser(User.Teacher, "Brandt", "Willi", "Willi.Brandt@beuth-hochschule.de")  
_createUser(User.Teacher, "Obama", "Barack", "Barack.Obama@beuth-hochschule.de")
```

Listing 5.4: Erzeugen der Beispielbenutzer in der Testsuite

Dann führen Sie die Testsuite aus. Sie sollte fehlerfrei durchlaufen. Beim erstmaligen Ausführen dieses Kommandos werden durch Maven ca. 32 Megabyte an Daten in das lokale Maven-Repository heruntergeladen. Dieser Vorgang kann einige Minuten dauern. Beim erneuten Start verkürzt sich diese Zeit auf wenige Sekunden, da die erforderlichen Daten von nun an aus dem lokalen Repository geladen werden. Für das Ausführen der Testsuite wie für alle Maven-Kommandos gehen Sie in das Projektwurzelverzeichnis, in dem die Datei `pom.xml` steht und beauftragen Maven mit dem Test:

```
$ cd liftbuchcode  
$ mvn test
```

Um den Nachrichtenversand als Lehrkraft „Knabe“ ausprobieren zu können, erzeugt die Testsuite auch schon 6 Abonnements bei der Lehrkraft „Knabe“:

¹⁷ <http://www.h2database.com/>

```
//Datei src/test/scala/com/lehrkraftnews/model/UserTest.scala, Ausschnitt subscribe:
_assureSubscriptionsAtKnabe("Knabe", "Heinrich", "Triel", "Merkel", "Brandt", "Obama")
```

Listing 5.5: Erzeugen der Beispiellabonnements in der Testsuite

Alle bei Kompilation und Test erzeugten Dateien werden im Unterverzeichnis `target` abgelegt. Wenn einmal unerklärliche Fehler auftreten sollten, ist es sinnvoll, mittels Kommando `mvn clean` dieses Verzeichnis zu löschen und die Testsuite erneut durchzuführen. Die H2-Datenbank wird im Verzeichnis `target/database` angelegt. Das Protokoll der Testtreiber-meldungen finden Sie in `target/lehrkraftnews.log`.

Start der Applikation

Anschließend kann der Jetty-Webserver im Verzeichnis `liftbuchcode` mit dem folgendem Befehl gestartet werden.

```
$ mvn jetty:run
```

Beim erstmaligen Start der Anwendung erscheinen auf der Konsole viele Warnungen der Art:

```
[WARNING] .../src/main/webapp/scripts/jquery-ui.min.js:line -1:column -1:
Try to use a single 'var' statement per scope.
ui[k].prototype.for( ----> var <--- j in n){m.plugins[j
```

Dabei handelt es sich um Warnmeldungen der Bibliothek *YUI-Compressor*, die genutzt wird, um JavaScript-Code zu komprimieren und seine Effizienz zu erhöhen. Die Warnmeldungen betreffen jQuery-Bibliotheken¹⁸ für die Umsetzung von Anzeigeffekten und signalisieren die Möglichkeit, deren Programmcode effizienter zu gestalten.

Wenn die Meldung `[INFO] Starting scanner at interval of 5 seconds.` im Konsolenfenster erscheint, ist der Jetty-Webserver betriebsbereit. Das Protokoll der Lift- und Anwendungsmeldungen finden Sie in `target/lehrkraftnews.log`.¹⁹

Der Server kann durch die Tastenkombination *Strg+C* heruntergefahren werden werden. Alternativ kann er mit `mvn jetty:stop` in einem zweiten Konsolenfenster gestoppt werden.

¹⁸ jQuery ist eine freie JavaScript-Bibliothek, die hauptsächlich Funktionalitäten für die DOM-Manipulation innerhalb von HTML-Seiten zur Verfügung stellt.

¹⁹ Zur Konfiguration des Loggings siehe Anhang A.2.

Um den verwendeten Port (8080) zu verändern, kann in der Datei `pom.xml` für das Jetty-Plugin ein Connector definiert werden. Mithilfe des `<port/>`-Elements kann dort der Port festgelegt werden.

```
<build>
...
<plugins>
...
<plugin>
<groupId>org.mortbay.jetty</groupId>
<artifactId>maven-jetty-plugin</artifactId>
<configuration>
<connectors>
<connector implementation="org.mortbay.jetty.nio.SelectChannelConnector">
<port>8180</port>
</connector>
</connectors>
<stopPort>9966</stopPort>
<stopKey>foo</stopKey>
<contextPath></contextPath>
<scanIntervalSeconds>6</scanIntervalSeconds>
</configuration>
</plugin>
...
</plugins>
...
</build>
```

Listing 5.6: Jetty-Plugin mit manuell definiertem Connector für Port 8180 in `pom.xml`

5.2.3. Bedienung

Beim Starten der Anwendung wird die Existenz eines Benutzers mit Administrator-Rechten in der Datenbank geprüft. Ist keiner vorhanden (z.B. beim erstmaligen Start der Anwendung oder wenn der Administrator den eigenen Benutzer gelöscht hat) wird ein solcher Benutzer angelegt.

Die Login-Daten für diesen Nutzer lauten:

- E-Mail: `admin@admin.de`
- Passwort: `admin`

Die Login-Daten können nachträglich im eingeloggten Zustand unter dem Menüpunkt „Benutzer bearbeiten“ geändert werden. Wenn Sie zuvor die Testsuite durchgeführt haben, sind schon 6 weitere Beispielbenutzer erzeugt. Siehe Listing 5.4.

Administratoren können registrierten Benutzern Rollen zuweisen. Ein Klick auf den Menüpunkt „Admin“ im Hauptmenü führt zur Benutzerverwaltung:

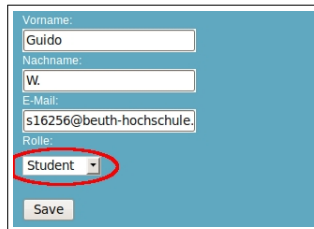


The screenshot shows a web interface for user management. At the top left, there is a link for "Benutzer Nachrichten". Below it is the title "Benutzerverwaltung". A table lists two users with columns for Name, Vorname, and E-mail. To the right of each row are two links: "Löschen" and "Bearbeiten". The "Bearbeiten" link for the second user is circled in red.

Name	Vorname	E-mail		
Grass	Günther	thomas@tz.dde	Löschen	Bearbeiten
W.	Guido	s16256@beuth-hochschule.de	Löschen	Bearbeiten

Abbildung 5.4.: Benutzerverwaltung

Von dort gelangt man über die „Bearbeiten“-Links zur Bearbeitungsansicht, wo die Benutzerrolle durch eine Auswahlliste festgelegt werden kann:



The screenshot shows a form for editing a user. It has input fields for "Vorname" (filled with "Guido"), "Nachname" (filled with "W."), and "E-Mail" (filled with "s16256@beuth-hochschule."). Below these is a dropdown menu for "Rolle" with "Student" selected and circled in red. A "Save" button is at the bottom.

Abbildung 5.5.: Bearbeitungsansicht

Um den Registrierungsprozess zu erleichtern, wurde die E-Mail-Validierung deaktiviert. Ihre Aktivierung wird im Abschnitt 7.9.2 erklärt.

Die Beschreibung der Anwendung in Abschnitt 5.1 liefert einen Überblick über die Funktionalitäten. Durch die aussagekräftigen Menüpunkte und die generierten Rückmeldungen wurde eine intuitive Bedienung der Anwendung angestrebt, weshalb mit Rücksicht auf den Umfang des Buches auf eine umfassendere Bedienungsanleitung verzichtet wird.

6. Test-Infrastruktur und Testorientierte Dokumentation

In diesem Kapitel wird beschrieben, wie eine *Test-Infrastruktur* in einem Lift/Scala-Projekt aufgesetzt wird, und wie diese für *Testgesteuerte Entwicklung* und *Testorientierte Dokumentation* genutzt werden kann.

Eine Test-Infrastruktur ist notwendig, um in einem Projekt mit Hilfe von reproduzierbaren Testfällen entwickeln und dokumentieren zu können.

6.1. Test-Infrastruktur

Unter einer *Test-Infrastruktur* wird eine Einrichtung eines Projektes derart verstanden, dass es einfach ist, Testfälle zu definieren und zusammengefasst als *Testsuite* reproduzierbar auszuführen. An die Ausführung können Entscheidungen (z.B. ob das Produkt gebaut wird) und Auswertungen (z.B. zur Testabdeckung) gekoppelt werden.

6.1.1. Framework-Auswahl, JUnit, SUnit, specs

Für die Einrichtung einer Test-Infrastruktur müssen ein oder mehrere Frameworks ausgewählt werden. Die Kriterien dafür sind

- Bekanntheit
- Lesbarkeit der Testfälle
- Integration in Maven
- Integration in Eclipse und IntelliJ IDEA

*JUnit*²⁰ ist sicherlich das am häufigsten eingesetzte Test-Framework auf der Java-Plattform. Die Kriterien sind wie folgt erfüllt.

²⁰ <http://www.junit.org/>

Bekanntheit	Höchstmögliche
Lesbarkeit	Etwas umständliche Notation <code>assertEquals(expected, actual)</code>
Maven-Integration	Sehr gut einschließlich Testabdeckungstool Cobertura
IDE-Integration	So gut, dass JUnit 4 auf eigene Testrunner verzichtet.

SUnit ist Bestandteil des Scala API und offensichtlich ein Nachbau von JUnit. Es ist daher ein naheliegender Kandidat. Es ist jedoch seit Scala 2.7.2 *Deprecated*. Anscheinend wurden unabhängig vom Scala API bessere Test-Frameworks entwickelt. In der SUnit-Dokumentation selbst werden *ScalaTest*, *ScalaCheck* und *Specs* genannt.

*specs*²¹ ist inspiriert durch Ruby's *RSpec*. Es ist verbunden mit dem Stil des *Behaviour-Driven-Design* (BDD). Die darin beschriebenen Testfälle lassen sich leicht lesen angenähert an englische Sätze. Dies wird jedoch nach Meinung des Autors erkauft durch eine Unmenge von Methodennamen der specs-DSL, die alle gelernt werden müssen, wenn man Testfälle schreiben will. Die Kriterien werden wie folgt bewertet.

Bekanntheit	Mittel, da der BDD-Pionier in der Scala-Welt
Lesbarkeit	Flüssig lesbare Testfälle mit verbal herausgearbeiteten Erwartungen
Maven-Integration	Unbekannt
IDE-Integration	Unbekannt

6.1.2. ScalaTest

*ScalaTest*²² ist ein umfassendes Test-Framework, welches verschiedene Ansätze integriert. Man kann damit Testfälle im Stile von JUnit, specs und weiteren Stilen schreiben. Im Scala-Buch von Odersky et al. [OdSV08] ist ihm ein eigenes Kapitel gewidmet. Die Kriterien werden wie folgt bewertet.

Bekanntheit	Groß, da im quasi offiziellen Scala-Buch von Odersky u.a. empfohlen.
Lesbarkeit	Einfach lesbare Assertions mit <code>assert(actual===expected)</code>
Maven-Integration	Gut bei Benutzung der JUnit-Kompatibilitätsklassen
IDE-Integration	Gut bei Benutzung der JUnit-Kompatibilitätsklassen

²¹ <http://code.google.com/p/specs/>

²² <http://www.scalatest.org/>

6.1.3. Nutzung von ScalaTest/JUnit in Maven und der IDE

In der Beispielapplikation und diesem Buch wird eine Test-Infrastruktur verwendet, die nur wenig von der gewohnten Nutzung von JUnit in Java-Projekten abweicht. Es wird ScalaTest mit JUnit-artigen Testfällen eingesetzt.

ScalaTest wurde ausgewählt, weil es das bekannteste Test-Framework in der Scala-Welt ist. Die Entscheidung für JUnit-artige Testfälle wurde getroffen, da erstens diese für Programmierer knapper das Wesentliche ausdrücken als es BDD-Specs tun. Zweitens ist nur bei Testfällen, die nach außen wie JUnit-Testfälle aussehen, die bruchlose Integration der Testsuite in die verwendete Werkzeuglandschaft aus Maven, Surefire, Cobertura und die IDE gewährleistet.

Andersgeartete ScalaTest-Testfälle lassen sich zwar auch in einer IDE als Applikation ausführen, werden aber nicht von der IDE oder von Surefire/Maven paketweise eingesammelt. Auch die einfache Navigation vom Stacktrace eines fehlgeschlagenen Testfalls zum Quellcode des Testfalls oder des Testlings funktioniert nur bei JUnit-artigen Testfällen.

6.2. Testgetriebene Entwicklung

Das mit *eXtreme Programming* [XP] populär gewordene Konzept der *Testgetriebenen Entwicklung* [TDD] sieht vor, dass für jede Anforderung an ein Programm zunächst ein Testfall geschrieben wird, der diese Anforderung abprüft. Erst danach versucht man, den Testfall durch eine geeignete Implementierung zu befriedigen. Durch dieses iterative Vorgehen erhält man eine umfangreiche *Testsuite* (= Testfallsammlung) quasi nebenbei.

Von den vielfältigen Mitteln, die uns ScalaTest zur Verfügung stellt, werden wir nur wenige verwenden, die hier kurz erklärt werden. Wir benutzen ScalaTest-Testfälle, die nach außen hin wie ein JUnit-Testfall aussehen. Eine Testklasse muss `org.scalatest.junit.JUnitSuite` beerben, um die ScalaTest-Assertions zur Verfügung zu stellen und sowohl als JUnit- als auch als ScalaTest-Suite aufrufbar zu sein.

Wir drucken hier in Listing 6.1 einen aus der `ScalaTest-ExampleSuite` abgeleiteten, vereinfachten Testfall ab. Er überprüft in Methode `verifyEasy` mittels JUnit-Mitteln und in Methode `verifyFun` mittels ScalaTest-Mitteln, dass der `StringBuilder`-Inhalt korrekt zusammgebaut wird.

In diesem Buch werden wir jedoch nur folgende ScalaTest-Mittel verwenden:

- `assert(actual === expected)`, da diese Art der Assertion lesbarer ist und bessere Meldungen als `assertEquals` von JUnit gibt.
- `intercept[exceptionClass]` zum Abprüfen auf erwartete Ausnahmen, da dies mit weniger Schreibaufwand verbunden ist als die JUnit-Entsprechung mit `try-fail-catch`.

```

package platform

import org.scalatest.junit.JUnitSuite
import org.junit.Assert._
import org.junit.Test
import org.junit.Before
import _root_.scala.collection.mutable.ListBuffer

/** Example JUnit-compatible test suite using ScalaTest */
class ExampleTest extends JUnitSuite {

  var sb: StringBuilder = _

  @Before def initialize() {
    sb = new StringBuilder("ScalaTest_is_")
  }

  @Test def verifyEasy() { // Uses JUnit-style assertions
    sb.append("easy!")
    assertEquals("ScalaTest_is_easy!", sb.toString)
    try {
      "verbose".charAt(-1)
      fail("expected:_StringIndexOutOfBoundsException")
    } catch {
      case expected: StringIndexOutOfBoundsException =>
    }
  }

  @Test def verifyFun() { // Uses ScalaTest assertions
    sb.append("fun!")
    assert(sb.toString === "ScalaTest_is_fun!")
    intercept[StringIndexOutOfBoundsException] {
      "concise".charAt(-1)
    }
  }
}

```

Listing 6.1: `src/test/scala/platform/ExampleTest.scala`

6.3. Testorientierte Dokumentation

Dokumente, in denen Programmiersprachen oder Programmierschnittstellen erklärt werden, müssen eine Vielzahl von Quellcodeausschnitten gemischt mit Text enthalten. Traditionellerweise werden dazu Programmbeispiele erstellt und daraus Ausschnitte manuell in den Text kopiert. Dieser Ansatz ist sehr änderungsfeindlich, da die Korrektheit der Codeschnipsel nur zum Zeitpunkt der Einfügung gewährleistet ist.

6.3.1. Extraktion von Codeschnipseln

In diesem Buch wird statt dessen ein automatisierter Ansatz verfolgt. Die Codeausschnitte in den Programmbeispielen werden in Kommentare mit `BEGIN(name)` und `END(name)` eingerahmt. Das eigene Werkzeug *ScalaSnipper* extrahiert daraus Ausschnittdateien, die im Listingformat in den Text inkludiert werden.

6.3.2. Testfälle als Programmbeispiele

Die neuartige Strategie der *Testorientierten Dokumentation* geht noch einen Schritt darüber hinaus. Traditionellerweise werden in einem Codeausschnitt gewisse Berechnungen vorgenommen und am Ende steht eine `println`-Anweisung, die das berechnete Ergebnis ausgibt. Siehe Listing 6.2, in dem das Funktionieren der vordefinierten Funktion `Math.sqrt` illustriert wird.

```
//Datei src/test/scala/platform/PlatformTest.scala, Ausschnitt sqrtTraditionell:  
val result = Math.sqrt(25)  
println(result) //gibt aus: 5
```

Listing 6.2: Traditionelles Codebeispiel mit Ausgabe

Automatisierte Testfälle haben sich in der Entwicklung sehr bewährt, weil sie zu jedem Zeitpunkt wiederholbar sind und keiner manuellen Überprüfung der Korrektheit ihres Ergebnisses bedürfen. Diesen Ansatz übertragen wir auf die Erstellung von Programmbeispielen. Statt das errechnete Ergebnis mit `println` auszugeben und das erwartete Ergebnis im Kommentar oder Text daneben zu schreiben, wird das Programmbeispiel um eine Zusicherung erweitert, in der die Gleichheit des errechneten mit dem erwarteten Ergebnis überprüft wird. Dadurch ist ebenfalls das erwartete Ergebnis im Text abgedruckt. Der Vorteil ist aber, dass die Korrektheit des Programmbeispiels durch die Einbettung in eine Testsuite garantiert ist, auch nach häufiger Änderung an ihm. Im Listing 6.3 ist das vorige Programmbeispiel als Testfall ausgedrückt.

```
//Datei src/test/scala/platform/PlatformTest.scala, Ausschnitt sqrtAlsTestfall:
```

```
val result = Math.sqrt(25)
```

```
assert(result == 5)
```

Listing 6.3: Das sqrt-Code-Beispiel umformuliert als Testfall

Die einzige Schwierigkeit dabei ist, dass diese Art von Dokumentation noch unüblich ist. Der Leser wird sich jedoch schnell an die Formulierung automatisierter Testfälle gewöhnen und soll motiviert werden, die Qualität seines Programmcodes durch Tests zu steigern. Programmbeispiele werden in diesem Buch also wann immer möglich als Testfall formuliert. Dadurch bietet dieses Werk eine ungewöhnlich hohe Korrektheit der Codeausschnitte.

7. Das Mapper-Framework

7.1. Grundsätzliches

7.1.1. Mapper/Record

Dieses Buch behandelt für die Umsetzung des Datenmodells ausschließlich das *Mapper*-Framework. Es ist nur auf relationale Datenbanken ausgerichtet. Das Mapper-Framework ist bewährt seit den Anfängen von Lift und unterstützt schon in der Standardauslieferung eine große Anzahl von relationalen Datenbanksystemen: Apache Derby, MySQL, MaxDB, PostgreSQL, H2 Database, Microsoft SQL Server, Sybase ASE und Oracle.

Aufgrund der Kompatibilität zu Java ist für relationale Datenbanken auch die Verwendung der Java Persistence API (JPA) in Lift möglich [ChDW09].

Bei dem sich zum Zeitpunkt der Entstehung des Buches noch in der Entwicklung befindlichen *Record*-Framework handelt es sich um eine Überarbeitung des Mapper-Frameworks mit dem Ziel, dieses auch für die aufkommenden NoSQL-Datenbanken zu öffnen. Es soll so allgemein formuliert sein, dass die benutzte Datenbankart (SQL/NoSQL) ausgetauscht werden kann. Es soll in zukünftigen Versionen von Lift zum Standard werden. Das Mapper-Framework soll dabei jedoch nicht sofort durch Record abgelöst werden, vielmehr werden beide Frameworks vorerst parallel existieren [GoGr09a]. Record-Implementierungen existieren zur Zeit nur für die NoSQL-Datenbanken CouchDB und MongoDB.

7.2. Integration des Mapper-Moduls in das Projekt

Mit dem Mapper-Framework lässt sich eine typsichere objektrelationale Abbildung eines Datenmodells erstellen. Beim Mapper-Framework handelt es sich um ein eigenständiges Lift-Modul. Sollte die Abhängigkeit vom Artefakt `lift-mapper` beim Erstellen des Projekts mit Maven nicht schon eingetragen worden sein, muss sie in der Datei `pom.xml` hinzugefügt werden. Wegen der Umstellung von Scala 2.7.7 auf Scala 2.8 muss im Lift-Artefaktnamen

die verwendete Scala-Version angegeben werden. Damit alle Lift-Artefakte in derselben Version angefordert werden, werden in der POM-Datei die *Properties* `scala.version` und `lift.version` definiert, auf die später Bezug genommen werden kann. Als Referenz werden im Listing 7.1 alle Property-Definitionen unterhalb von `<project>` wieder gegeben.

```
<!-- Datei pom.xml, Ausschnitt properties: -->
<properties>
  <!-- BEGIN(versions) -->
  <scala.version>2.8.0</scala.version>
  <lift.version>2.2</lift.version>
  <!-- END(versions) -->
  <!-- Common plugin settings -->
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>
    ${project.build.sourceEncoding}
  </project.reporting.outputEncoding>
</properties>
```

Listing 7.1: Property-Definitionen in pom.xml

Durch Verwendung der Maven-Property `${scala.version}` in dem Namen der angeforderten Lift-Artefakte unterhalb von `<dependencies>` wird Mapper in der zur Scala-Version passenden Variante angefordert (Listing 7.2).

```
<!-- Datei pom.xml, Ausschnitt mapper: -->
<dependency>
  <groupId>net.liftweb</groupId>
  <artifactId>lift-mapper_${scala.version}</artifactId>
  <version>${lift.version}</version>
</dependency>
```

Listing 7.2: Mapper-Abhängigkeit in pom.xml

7.3. Konfiguration des Datenbankzugriffs

Die Wahl des Datenbanktreibers und dessen Konfiguration erfolgt in der Datei `Boot.scala`, die sich in dem Paket `bootstrap.liftweb` befindet. Dazu wird ein von `ConnectionManager` erbenendes Objekt `DBVendor` definiert. Dessen Basistrait ist nachfolgend abgedruckt:

```
trait ConnectionManager {
  def newConnection(name: ConnectionIdentifier): Box[Connection]
  def releaseConnection(conn: Connection)
  def newSuperConnection(name: ConnectionIdentifier): Box[SuperConnection] = Empty
```

}

Listing 7.3: Trait `net.liftweb.mapper.ConnectionManager`

DBVendor muss also die beiden Methoden `newConnection` und `releaseConnection` implementieren. `newConnection` liefert eine `Box`²³ mit einem `Connection`-Objekt zurück. In DBVendor folgen wir dem Muster des Lift-Wiki bezüglich des *Connection Pooling*. Da der Code zu vielen Zeilen unverändert aus [LiftWikiDBC] übernommen wurde, wird er hier nicht abgedruckt.

Der meist projektspezifisch anzupassende Teil findet sich in der Methode `DBVendor.createOne`, die bei Bedarf von `newConnection` aufgerufen wird. Daher ist der Anfang des Objekts mit dieser Methode für eine H2-Datenbank im Listing 7.4 dargestellt. Sie ist unter der Adresse „localhost“ ohne Benutzernamen und Passwort zu erreichen und wird im relativ zur Projektwurzel angegebenen Verzeichnis `target/database/H2` abgelegt. Dies vermeidet eine zufällige Versionierung. Beim Löschen aller generierten Dateien mittels `mvn clean` wird dadurch auch diese Datenbank gelöscht. Dies ist für die Entwicklungs- und Testphase durchaus sinnvoll.

Kann keine Verbindung hergestellt werden, wirft `createOne` eine mit dem Treibernamen und der Verbindungs-URL parametrisierte Ausnahme.²⁴

```
// Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt DBVendor:
object DBVendor extends ConnectionManager {

  private def createOne: Box[Connection] = {
    val url = "jdbc:h2:target/database/H2"
    //Für Compile-Zeit-Existenzprüfung der Treiberklasse:
    val driverClass = classOf[org.h2.Driver]
    val cName = driverClass.getName
    try {
      Class.forName(cName)
      val connection = DriverManager.getConnection(url)
      connection.setAutoCommit(false) //for using transactions
      Full(connection)
    } catch {
      case e: Exception =>
        throw new Failure("Cannot_create_DB_connection_with_driver_{0}_for_URL_{1}",
          e, cName, url
        )
    }
  }
} //createOne
```

Listing 7.4: Konfiguration einer H2-Datenbank in der Datei `Boot.scala`

²³ siehe Anhang A

²⁴ In den üblichen Lift-Beispielen wird abweichend eine diagnoseschwächere `Box` vom Typ `Empty` geliefert.

Damit alle Datenbankänderungen einer Anfrage in einer Transaktion²⁵ ausgeführt werden, wird in `newConnection` für jede neu erzeugte `Connection` mittels `connection.setAutoCommit(false)` das automatische `Commit` ausgeschaltet. Stattdessen wird in der `boot`-Methode mittels `S.addAround` jeder Request in eine Transaktion eingefasst. Es gibt eine Methode `buildLoanWrapper` in dem Objekt `DB`, die eine entsprechende Behandlungsfunktion generiert:

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt transaction:
//For making each request a database transaction:
S.addAround(DB.buildLoanWrapper(List(DefaultConnectionIdentifier)))
```

Listing 7.5: Einstellung des zentralen Transaktionsmanagements in Datei `Boot.scala`

Um die in `DBVendor` definierte Verbindungsverwaltung für das Mapper-Framework nutzen zu können, muss sie in der `boot`-Methode mithilfe von `DB.defineConnectionManager` registriert werden:

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt connection:
if (!DB.jndiJdbcConnAvailable_?){
  DB.defineConnectionManager(DefaultConnectionIdentifier, DBVendor)
}
```

Listing 7.6: Konfiguration des Database Connection Managers in der Datei `Boot.scala`

Der H2-Datenbanktreiber wird als Modul benötigt. Daher muss er in der Datei `pom.xml` als Abhängigkeit eingefügt werden:

```
<!-- Datei pom.xml, Ausschnitt h2database: -->
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.1.112</version>
</dependency>
```

Listing 7.7: Anforderung des H2-Datenbanktreibers in `pom.xml`

Ein Wechsel der verwendeten Datenbank erweist sich als einfach. Um beispielsweise PostgreSQL als Datenbank zu benutzen, muss der entsprechende Treiber in der Datei `pom.xml` als Modul geladen, und die Methode `newConnection` in `DBVendor` bezüglich Treiberklasse, Datenbank-URL und Zugangsdaten angepasst werden.

```
<dependency>
  <groupId>postgresql</groupId>
  <artifactId>postgresql</artifactId>
```

²⁵ [http://de.wikipedia.org/wiki/Transaktion_\(Informatik\)](http://de.wikipedia.org/wiki/Transaktion_(Informatik))

```
<version>8.1-404.jdbc3</version>
</dependency>
```

Listing 7.8: Anforderung des PostgreSQL-Datenbanktreibers in pom.xml

```
...
val driverClass = classOf[org.postgresql.Driver]
Class.forName(driverClass.getName)
Full(DriverManager.getConnection("jdbc:postgresql://localhost/lehrkraftnews", "root", ""))
...
```

Listing 7.9: Konfiguration einer PostgreSQL-Datenbank in der Datei Boot.scala

7.4. Erstellung von Modell-Klassen

Standardmäßig ist das Paket `model` für die Platzierung der Datenmodell-Klassen vorgesehen. Anhand der Klasse `com.lehrkraftnews.model.News` wird im folgenden erläutert, wie eine Modell-Klasse in Lift definiert wird, um mit Mapper auf eine Datenbanktabelle abgebildet zu werden. Die folgenden Abschnitte beziehen sich auf diese Klasse, sie ist im Listing 7.10 abgebildet:

```
package com.lehrkraftnews.model

import net.liftweb.mapper._
import java.text.SimpleDateFormat
import java.util.{Locale, Date}
import net.liftweb.http._
import net.liftweb.util._
import net.liftweb.common._
import scala.xml.{NodeSeq, Text}

/**Meta (Kompagnion)–Objekt für die News–Klasse. Enthält instanzübergreifende Einstellungen.
 * @author Thomas Fiedler */
//BEGIN(crud)
object News extends News with LongKeyedMetaMapper[News] with CRUDify[Long, News]
//END(crud)
{
  /**Name der genutzten Tabelle in der Datenbank*/
  override def dbTableName = "nachricht"
  /**Lokaler URI–Präfix dennoch auf Englisch, z.B. in: /news/edit/1 */
  //override def calcPrefix = List("news")
  /**Anordnung der Eingabefelder in automatisch generierten Formularen (CRUDify)*/
  override def fieldOrder = List(content, expirationDate, byUserId)
  /**Name des Menüpunktes für die Ansicht aller Objekte für CRUDify–Seiten*/
  override def showAllMenuName = "Nachrichten"
```

```

/**Name des Menüpunktes für das Erstellen eines neuen Objekts auf CRUDify-Seiten*/
override def createMenuName = "Nachricht_erstellen"

//BEGIN(pageWrapper)
/**Einbettung aller News-CRUD-Seiten in das default-Template mit Admin-Menü und Titelanzeige
* eines "Erstellen"-Links*/
override def pageWrapper (body: NodeSeq) = {
  <lift:surround with="default" at="content">
    <lift:embed what="admin_menu"/>
    <h1><lift:Menu.title/></h1>

    <div id="formBox">
      {body}
    </div>
  </lift:surround>
}
//END(pageWrapper)
}

/**Beschreibt eine Nachrichten-Instanz
* @author Thomas Fiedler */
//BEGIN(crudModify)
class News extends LongKeyedMapper[News] with IdPK {

  /**Beschreibt Datenfeld für den Autor einer Nachricht als Fremdschlüssel für Relation zu User-Objekten*/
  object byUserId extends MappedLongForeignKey(this, User){

    /**Genutzter Spaltenname in der DB-Tabelle*/
    override def dbColumnName = "betrifft_person_id"

    /**Darstellung des Feldes auf CRUD-Seiten. Anstelle der Id wird Nachname und Vorname des Autors
    * angezeigt bzw. "k.A." für "keine Angabe", wenn es zu dieser User-Id keinen User gibt. */
    override def asHtml = Text(User.find(this).map(_fullCommaName).openOr("k.A. "))

    /**Name des Datenfeldes für CRUD-Seiten*/
    override def displayName = "Lehrkraft"

    /**Namen-Auswahlliste für CRUD-Seiten*/
    override def validSelectValues: Box[List[(Long, String)]] =
      Full(User.allTeachers.map(u => (u.id.is, u.lastName.is)))
  }
  //END(crudModify)

  // BEGIN(content)
  /**Beschreibt Datenfeld für den Inhalt einer Nachricht*/
  object content extends MappedTextarea(this, 2000){
    /**Genutzter Spaltenname in der DB-Tabelle*/

```



```

override def dbColumnName = "inhalt"

/**Liste der durchzuführenden Validationen*/
override def validations = notEmpty _ :: Nil

/**Definition der Validationsbedingung "Feld darf nicht leer sein"*/
def notEmpty(s: String) = {
  if (s.trim.length == 0)
    List(FieldError(this, Text("Feld_\\"Inhalt\"_darf_nicht_leer_sein")))
  else
    List[FieldError]()
}
}
// END(content)

/**Datumsfeld für die Gültigkeit der Nachricht */
object expirationDate extends MappedDateTime(this) {
/**Genutzter Spaltenname*/
override def dbColumnName = "gueltig_bis"

/**Zu prüfende Validationsbedingungen*/
override def validations = isValid _ :: Nil

/**Validation für ein Kalenderdatum.*/
def isValid(date: Date): List[FieldError] = {
  if (date == null) {
    return List(FieldError(this, Text("Datum_fehlt")))
  }
  /**Frühestes erlaubtes Datum*/
  val earliestDate = new SimpleDateFormat("dd.MM.yyyy").parse("01.01.2009")
  /**Spätestes erlaubtes Datum*/
  val latestDate = new SimpleDateFormat("dd.MM.yyyy").parse("01.01.2020")
  if (date.before(earliestDate) || date.after(latestDate)) {
    return List(FieldError(this, Text(
      "Ungültiges_Datum:_Datum_bereits_vergangen_oder_zu_weit_in_der_Zukunft")))
  }
  Nil
}
}

/**Liefert das Meta-Objekt zur eigenen Modellklasse.*/
def getSingleton = News

/**Liefert das Gültig-Bis-Datum als Zeichenkette im Format dd.MM.yyyy */
def getGermanDateString : String = {
  val sdfGerman: SimpleDateFormat = new SimpleDateFormat("dd.MM.yyyy", Locale.ENGLISH);
  if (this.expirationDate.is != null) {
    sdfGerman.format(this.expirationDate.is)
  } else ""
}
}

```

```

/**Hilfsfunktion zum Auslesen aller abonnierten Nachrichten eines Benutzers aus der Datenbank
 * unter Nutzung eines nativen SQL-Statements.
 *
 * @param id Id des Benutzers
 * @param offset Offset-Wert (Paginierung)
 * @param maxRows Menge der zu beschaffenen Nachrichten (Paginierung)
 * @return Liste aller Abonnierten Nachrichten des Benutzers mit der Id id
 */
//BEGIN(forSubscriber)
def newsBySubscriber(id: Long, offset: Int, maxrows: Int): List[News] =
News.findAllByPreparedStatement({ superconn => {
  val preparedStatement = superconn.connection.prepareStatement(
    "SELECT n.* FROM nachricht AS n LEFT JOIN abonnement AS a " +
    "ON (n.betrifft_person_id = a.nachrichtenquelle_person_id)" +
    "WHERE a.interessent_person_id = ? ORDER BY n.gueltig_bis DESC LIMIT ?,?"
  )
  preparedStatement.setInt(1, id.toInt)
  preparedStatement.setInt(2, offset.toInt)
  preparedStatement.setInt(3, maxrows.toInt)
  preparedStatement
}
//END(forSubscriber)

/* Gleiches Statement in PostgreSQL Syntax:
def newsBySubscriber(id : Long, offset: Int, maxrows: Int) : List[News] =
News.findAllByPreparedStatement({ superconn => {
  val preparedStatement = superconn.connection.prepareStatement(
    "SELECT n.* FROM nachricht AS n LEFT JOIN abonnement AS a " +
    "ON (n.betrifft_person_id = a.nachrichtenquelle_person_id)" +
    "WHERE a.interessent_person_id = ? ORDER BY n.gueltig_bis DESC LIMIT ? OFFSET ?"
  )
  preparedStatement.setInt(1, id.toInt);
  preparedStatement.setInt(2, offset.toInt);
  preparedStatement.setInt(3, maxrows.toInt);
  preparedStatement;
}
*/
})

/**Prüft ob eine Nachricht von einem Benutzer erstellt wurde
 * @param userId Id des Benutzers
 * @return Angabe ob die Nachricht vom Benutzer erstellt wurde
 */
def belongsToSubscribed(userId : Long): Boolean = {
  User.find(userId).openOr(User).subscriptions.exists(_sourceId == this.userId)
}

```

```
}
```

Listing 7.10: Die Modell-Klasse News

7.4.1. Mapper- und MetaMapper-Trait

Zur Definition einer Datenstruktur gehört neben der eigentlichen Modellklasse, welche die Funktionalität für die einzelnen Instanzen eines Modells bereitstellt, ein Begleiter-Objekt (im Folgenden Meta-Objekt genannt), also ein *Singleton-Objekt*, welches den gleichen Namen wie seine zugehörige Klasse trägt. Dieses Meta-Objekt beinhaltet instanzübergreifende Felder und Methoden. Die Modellklasse erbt vom Mapper-Trait, das Meta-Objekt vom MetaMapper-Trait. Die genannten Traits besitzen wiederum Unter-Traits, um die Entitäten mit speziellen Eigenschaften wie zum Beispiel dem Vorhandensein von Primärschlüsseln auszustatten. Das Meta-Objekt muss der Klasse über die Implementierung der Funktion `getSingleton` zugeordnet werden.

7.4.2. Primärschlüssel mit LongKeyedMapper und IdPK

Die Klasse `News` erbt von zwei Traits: `LongKeyedMapper` und `IdPK`. `LongKeyedMapper` sorgt dafür, dass das `News`-Modell einen Primärschlüssel im Long-Format erhält. Er erfordert, dass seine abstrakte Methode `primaryKeyField` implementiert wird. Dies leistet der `IdPK`-Trait. Zu beachten ist, dass der `IdPK`-Trait nur Primärschlüssel im Long-Format unterstützt. Wird ein anderes Format gewünscht, muss `primaryKeyField` selbst implementiert werden.

7.4.3. Festlegung des Tabellennamens

Um den Namen der benutzten Tabelle in der Datenbank festzulegen, kann die Methode `tableName` im Meta-Objekt überschrieben werden. Wird dies nicht getan, erhält die Tabelle den Namen der Klasse. Ist dieser identisch mit einem SQL-Schlüsselwort, erhält er die Endung `_t`.

7.4.4. Definition von Datenfeldern

Die einzelnen Datenfelder des Modells werden durch Singleton-Objekte definiert. Für die unterschiedlichen Datentypen innerhalb der Datenbank stehen entsprechende Traits zur Verfügung, von denen diese Objekte erben können. Diesen wird im Konstruktor standardmäßig die Modell-Klasse übergeben (mit `this`). Der verwendete Spaltenname in der Datenbank kann durch das Überschreiben der Methode `dbColumnName` festgelegt werden. Wird dies nicht getan, wird die Spalte mit dem Namen des Datenfeldes in der Klasse benutzt. Ist dieser identisch mit einem SQL-Schlüsselwort wird die Endung `_c` angehängt.

Die News-Klasse besitzt 3 Datenfelder:

byUserId Das Feld `byUserId` definiert einen Fremdschlüssel und verweist auf einen User, den Urheber der Nachricht. Zu diesem Zweck erbt das Objekt `byUserId` von `MappedLongForeignKey`. Als Parameter im Konstruktor `MappedLongForeignKey(this, User)` wird das Meta-Objekt der User-Klasse übergeben um die Beziehung zwischen News und User unter Verwendung des User-Primärschlüssels herzustellen. Der Name der verwendeten Spalte in der Datenbanktabelle wird auf `betrifft_person_id` geändert.

content Das Feld `content` soll den Inhalt einer Nachricht enthalten. Daher erbt es vom `MappedTextarea-Trait`. Der Parameter `2000` im Konstruktor begrenzt den Text auf eine Länge von 2000 Zeichen. Der Name der verwendeten Spalte in der Datenbanktabelle wird auf `inhalt` geändert. Zusätzlich werden durch das Überschreiben der Methode `validations` Regeln für die Validierung des Feldes festgelegt. Dies wird im Abschnitt 7.4.6 genauer beschrieben.

expirationDate Das Feld `expirationDate` beinhaltet das Datum, bis zu welchem eine Nachricht gültig ist. Für Datumsfelder steht der Trait `MappedDateTime` zur Verfügung. Der Name der verwendeten Spalte für dieses Feld in der Datenbanktabelle wird auf `gueltig_bis` geändert. Dieses Feld wird ebenfalls validiert.

7.4.5. Hinzufügen von Hilfsfunktionen

Um wiederkehrende Abfragen zu vereinfachen, können Hilfsfunktionen hinzugefügt werden. Die Funktion `getGermanDateString` liefert das Datum aus dem Feld `expirationDate` im

deutschen Datumsformat zurück. Die Funktion `belongsToSubscribed` prüft, ob eine Nachricht zu den abonnierten Nachrichten eines Benutzers gehört und liefert einen entsprechenden Boolean-Wert zurück.

7.4.6. Validierung von Datenfeldern

Beim Speichern von Datenfeldern können Validierungsbedingungen festgelegt werden. Werden diese nicht erfüllt, wird der Speichervorgang abgebrochen und der Benutzer erhält eine entsprechende Benachrichtigung.

Validierungsbedingungen werden innerhalb der Felddefinitionen formuliert. Dabei handelt es sich um Funktionen, die den Wert des Feldes als Parameter erhalten und eine Liste mit `FieldError`-Objekten zurückgeben. Diese `FieldError`-Objekte besitzen einen `NodeSeq`-Parameter, durch den beim Scheitern der Validierung eine geeignete Fehlermeldung ausgegeben werden kann. Wird eine leere Liste zurückgegeben, gilt die Validierung als erfolgreich.

Durch das Überschreiben der Methode `validations` können Validierungsfunktionen der Liste durchzuführender Validierungen hinzugefügt werden. Die Listenform ermöglicht die Aneinanderreihung mehrerer Validierungsbedingungen.

```
// Datei src/main/scala/com/lehrkraftnews/model/News.scala, Ausschnitt content:
/**Beschreibt Datenfeld für den Inhalt einer Nachricht*/
object content extends MappedTextarea(this, 2000){
  /**Genutzter Spaltenname in der DB-Tabelle*/
  override def dbColumnName = "inhalt"

  /**Liste der durchzuführenden Validationen*/
  override def validations = notEmpty _ :: Nil

  /**Definition der Validationsbedingung "Feld darf nicht leer sein"*/
  def notEmpty(s: String) = {
    if (s.trim.length == 0)
      List(FieldError(this, Text("Feld_\\"Inhalt\\"_darf_nicht_leer_sein")))
    else
      List[FieldError]()
  }
}
```

Listing 7.11: Validierung des Datenfeldes `content` in der Model-Klasse `News`

Listing 7.11 zeigt die Definition des Datenfeldes `content` aus der Modell-Klasse `News`. Zunächst wird die Funktion `notEmpty` definiert, die prüft, ob der Inhalt des Feldes leer ist.

Durch das Anhängen an die von `validations` zurückgegebene Liste wird `notEmpty` für die Validierung des Feldes `content` eingesetzt.

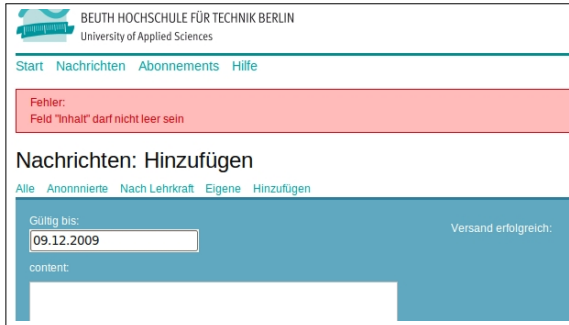


Abbildung 7.1.: Fehlermeldung bei gescheiterter Validierung

Die Ausgabe einer Validierungsfehlermeldung erfolgt als Error-Nachricht und wird durch das `Msgs`-Snippet ausgegeben. (siehe Abschnitt 8.4).

7.5. Objekt-Operationen

Für die grundlegenden Datenbankoperationen `Create` (Datensatz anlegen), `Read` (Datensatz lesen), `Update` (Datensatz aktualisieren) und `Delete` (Datensatz löschen) stehen verschiedene Methoden des `MetaMapper`-Traits zur Verfügung.

7.5.1. Erstellen und Speichern

Für das Erstellen einer neuen Instanz wird die Methode `create` des entsprechenden Meta-Objektes verwendet. Jedem einzelnen Feld kann ein Wert mit der `apply`-Methode bzw. deren Kurzschreibweise `feld(wert)` zugewiesen werden. Zum Speichern wird die `save`-Methode auf die erstellte Instanz angewendet. Im folgenden Beispiel wird ein Objekt der im Listing 7.10 definierten Klasse `News` erstellt. Seine Felder werden mit Werten gefüllt, das Feld `content` dabei mittels der `apply`-Methode, die anderen beiden Felder mit deren Kurzschreibweise. Anschließend wird das Objekt gespeichert.

```
// Datei src/test/scala/com/lehkräftenews/model/NewsTest.scala, Ausschnitt create1:
```

```
val message = News.create
message.userId(1)
message.content.apply("Inhalt_der_1._Nachricht")
val now = new java.util.Date
message.expirationDate(now)
message.save
```

Listing 7.12: Beispiel: create

Diese Befehle können auch verkettet aufgerufen werden. Dies ist die kurze und übliche Notation, mit der ein Objekt persistiert wird:

```
// Datei src/test/scala/com/lehkrainews/model/NewsTest.scala, Ausschnitt create2:
News.create.userId(2).content("Inhalt_der_2._Nachricht").expirationDate(new Date).save
```

Listing 7.13: Beispiel: verkettetes create

Es ist zu beachten, dass das `save` keine Validierung durchführt! Man sollte ihm also nur schon validierte Objekte übergeben.

7.5.2. Zugriff auf Datenfelder

Für den lesenden Zugriff auf die Datenfelder eines Objekts steht die `is`-Methode zur Verfügung. Wenn man den Feldwert vergleichen möchte, ist auch die überladene Methode `==` benutzbar. In folgendem Testfall werden zu jedem Feld des in Listing 7.12 erstellten Nachrichtenobjekts die mittels `is` geholten Feldwerte überprüft. Außerdem wird jedes Feld noch einmal direkt mit der Methode `==` überprüft.

```
// Datei src/test/scala/com/lehkrainews/model/NewsTest.scala, Ausschnitt fieldAccess:
assert(message.userId.is === 1)
assert(message.userId == 1)
assert(message.content.is === "Inhalt_der_1._Nachricht")
assert(message.content == "Inhalt_der_1._Nachricht")
assert(message.expirationDate.is === now)
assert(message.expirationDate == now)
```

Listing 7.14: Beispiel: Zugriff auf Datenfelder

7.5.3. Zugriff auf Objekt-Verbindungen

In der Klasse `News` in Listing 7.10 wurde durch das Feld `byUserId` eine Assoziation zu der Klasse `User` hergestellt. Die `obj`-Methode des Feldes liefert direkt das entsprechende `User`-Objekt in einer `Box`.²⁶ Im folgenden Beispiel wird zunächst eine Nachricht über den Admin-User erzeugt, dann auf den in der Nachricht enthaltenen User-Fremdschlüssel mittels `byUserId.is`, auf das verbundene User-Objekt mittels `byUserId.obj` zugegriffen und dann wird das verbundene User-Objekt noch einmal vereinfacht verglichen nur unter Verwendung des Feldnamens `byUserId`. Es gibt also eine Reihe von Notationen, um das über einen Fremdschlüssel identifizierte Objekt zu benutzen.

```
//Datei src/test/scala/com/lehkrkraftnews/model/NewsTest.scala, Ausschnitt verbindung:
//Holen des Admin-Benutzers:
val admin = User.findAll(
  By(User.lastName, "Admin")
)(0)
//Erzeugen einer Nachricht über diesen Benutzer:
val msg: News = News.create
msg.byUserId(admin).content("Admin_bereit").expirationDate(new Date).save
//Überprüfen der Fremdschlüssel-Id:
val id: Long = msg.byUserId.is
assert(id == admin.id.is)
//Überprüfen des verbundenen Objekts:
val userBox: Box[User] = msg.byUserId.obj
assert(userBox.open_! == admin)
//Verkürzte Schreibweise zum Überprüfen des verbundenen Objekts:
assert(userBox == admin)
//Man kann auch direkt das Fremdschlüsselfeld mit dem verbundenen Objekt vergleichen:
assert(msg.byUserId == admin)
```

Listing 7.15: Beispiel: Zugriff auf Objekt-Verbindungen

7.5.4. Aktualisieren

Für Updates von Datensätzen wird ebenfalls die `save`-Methode verwendet. Mapper erkennt anhand der intern genutzten Flags `saved_?` und `clean_?`, ob ein Objekt durch INSERT neu in die Datenbank eingefügt, oder durch UPDATE aktualisiert werden muss.

²⁶ Eine Erläuterung des `Box`-Konzepts erfolgt im Anhang. Das Verständnis dieses Konstruktes ist grundlegend für die Entwicklung mit Lift.

7.6. Datenbankabfragen

Während die Operationen zum Einfügen, Ändern und Löschen eines Datenmodellobjekts unproblematisch sind, tritt bei den Abfragen die ganze Komplexität eines Datenbanksystems zu Tage. Insbesondere eine typischere Lösung ist mit programmiersprachlichen Mitteln normalerweise äußerst schwer zu erreichen.

7.6.1. Methoden

Der `MetaMapper`-Trait stellt typischere Methoden für Datenbankabfragen zur Verfügung:

findAll Dieser Methode können beliebig viele Objekte des `QueryParam`-Traits übergeben werden (siehe Abschnitte 7.6.2 und 7.6.3). Sie liefert eine Liste mit Objekten aus der Datenbank zurück, welche die Eigenschaften besitzen, die durch die `Query`-Parameter gefordert wurden. Wird kein Parameter übergeben, werden alle Einträge zurückgeliefert. Folgendes Beispiel gibt den Inhalt aller in der Datenbank vorhandenen Nachrichten als Log-Nachricht aus.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findAll:  
News.findAll.foreach(msg => log.info("Inhalt:_" + msg.content.is))
```

Listing 7.16: Beispiel: findAll

count Arbeitet wie `findAll`, liefert aber anstelle einer Liste die Anzahl der gefundenen Einträge als Long-Wert.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt count:  
log.info("Anzahl_der_vorhandenen_Nachrichten:_" + News.count)
```

Listing 7.17: Beispiel: count

find Für Modell-Klassen, die von `KeyedMetaMapper` und seinen Unter-Traits erben, steht die Methode `find` zur Verfügung. Sie liefert das Objekt mit dem Primärschlüssel, der im `id`-Parameter übergeben wird, in einer `Box` zurück. Wird im folgenden Beispiel die Nachricht mit der Id 5 gefunden, wird ihr Inhalt ausgegeben, falls nicht, erscheint der Text „Nicht vorhanden“.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt find:
val inhalt = News.find(5).map(msg => "Inhalt:_" + msg.content.is) openOr "Nicht_vorhanden"
log.info(inhalt)
```

Listing 7.18: Beispiel: find

findAllByPreparedStatement Um Abfragen in nativem SQL zu formulieren kann die Methode `findAllByPreparedStatement` genutzt werden (siehe Abschnitt 7.6.4).

7.6.2. Vergleichs-Query-Parameter

Das Mapper Framework stellt verschiedene Query-Parameter bereit, mit denen sich typischer viele Elemente der SQL-Syntax darstellen lassen. In diesem Abschnitt werden die Vergleichs-Query-Parameter beschrieben, mit denen sich die Ergebnismenge einer Abfrage durch Vergleiche der Feldwerte mit Vorgaben einschränken lässt.

Feldwertvergleich

Für Feldwertgleichheit kann der `By`-Parameter verwendet werden. Er verhält sich wie der SQL-Operator „=“. Analog funktionieren `By_>`, `By_<` und `NotBy`. Sie agieren wie „>“, „<“ und „!“ in der SQL-Syntax. Das folgende Beispiel findet alle `News`-Objekte, deren Feld `byUserId` den Wert 5 hat, in Form einer `List[News]`.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findAllBy:
val newsOfUser5: List[News] = News.findAll(
  By(News.byUserId, 5)
)
```

Listing 7.19: Beispiel: By-Parameter

Feldwertvergleich mit Liste

Mit dem `ByList`-Parameter besteht die Möglichkeit, eine Anfrage mit ODER-verknüpften Vergleichen auszuführen, indem die möglichen Werte in einer Liste übergeben werden. Nachfolgendes Beispiel zeigt die Ausgabe der Inhalte aller Nachrichten, deren Urheber entweder die Id 6 oder die Id 12 besitzt. Zu beachten ist, dass der `Elemente`-Typ als generischer

Parameter für `List` hierbei mit angegeben werden muss, da die Literale `6` und `12` andernfalls eine `List[Int]` ergeben würden.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findallByList:
News.findAll(
  ByList(News.byUserId, List[Long](6, 12))
).foreach(msg => log.info("Inhalt:_" + msg.content))
```

Listing 7.20: Beispiel: `ByList`-Parameter mit Typangabe `Long`

Um die Angabe des `List`-Elementtyps zu sparen, könnte man auch die einzelnen `Ids` als `Long`-Literale angeben wie in folgendem Beispiel.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findallByList:
News.findAll(
  ByList(News.byUserId, List(6L, 12L))
).foreach(msg => log.info("Inhalt:_" + msg.content))
```

Listing 7.21: Beispiel: `ByList`-Parameter mit `Long`-Literalen

Stringvergleich mit Muster

Der `Like`-Parameter wird benutzt, um `String`-Feldwerte mit einem Muster zu vergleichen. Das „`%`“-Zeichen kann wie in der `SQL`-Syntax als Wildcard eingesetzt werden. Das nächste Beispiel gibt den Inhalt aller Nachrichten aus, in denen das Wort „Ausfall“ vorkommt.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findallLike:
News.findAll(
  Like(News.content, "%Ausfall%")
).foreach(msg => log.info("Inhalt:_" + msg.content))
```

Listing 7.22: Beispiel: `Like`-Parameter

Vergleich auf `NULL` bzw. Nicht-`NULL`

Der `NotNullRef`-Parameter ermöglicht es, nach Einträgen zu suchen, bei denen das angegebene Feld den Wert `NULL` hat. Analog dazu kann der `NotNullRef`-Parameter verwendet werden, um Einträge zu finden, bei denen der Feldwert nicht `NULL` ist. Im folgenden Beispiel wird der Inhalt aller Nachrichten ausgegeben, deren Gültig-Bis-Datum (`expirationDate`) den Wert `NULL` hat:

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findallNullref:
News.findAll(
  NullRef(News.expirationDate)
).foreach(msg => log.info("Inhalt:_" + msg.content))
```

Listing 7.23: Beispiel: NullRef-Parameter

UND-Verknüpfung mehrerer Query-Parameter

Werden mehrere Query-Parameter übergeben, werden die Suchkriterien UND-verknüpft auf die Suche angewendet. Im folgenden Beispiel wird der Inhalt aller Nachrichten ausgegeben, in denen das Wort „Ausfall“ vorkommt und deren Urheber die Id 6 oder 12 besitzt.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findallLikeByList:
News.findAll(
  Like(News.content, "%Ausfall%"),
  ByList(News.userId, List[Long](6, 12))
).foreach(msg => log.info("Inhalt:_" + msg.content))
```

Listing 7.24: Beispiel: UND-Verknüpfung mehrerer Query-Parameter

7.6.3. Steuer-Query-Parameter

Weiterer Einfluss auf Datenbankabfragen kann durch *Steuer-Query-Parameter* (controlling query parameters) genommen werden.

Sortieren lassen sich die Resultate einer Abfrage durch den *OrderBy-Parameter*. Die Sortierrichtung kann durch die Parameter *Ascending* (aufsteigend) und *Descending* (absteigend) bestimmt werden. Die Anzahl der Resultate lässt sich durch den *MaxRows-Parameter* begrenzen. Der *StartAt-Parameter* gibt die Möglichkeit, den Offset-Wert der Abfrage zu bestimmen, also die Anzahl der Einträge, die übersprungen werden, bevor die durch *MaxRows* definierte Anzahl der Einträge geliefert wird.

Das nächste Beispiel zeigt eine Kombination aus drei Steuer-Parametern und einem Vergleichs-Parameter. Von allen Einträgen, bei denen das Wort „Ausfall“ im Feld *content* vorkommt, werden die ersten zwei übersprungen und dann vier geliefert. Es wird also der Inhalt des 3. bis 6. Eintrags in alphabetisch aufsteigender Reihenfolge ausgegeben.

```
//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findallControl:
News.findAll(
  Like(News.content, "%Ausfall%"),
```

```

    OrderBy(News.content, Ascending),
    MaxRows(4),
    StartAt(2)
  ).foreach(msg => log.info("Inhalt:_" + msg.content))

```

Listing 7.25: Beispiel: Kombination aus Vergleichs- und Steuer-Parametern

Eine sinnvolle Anwendung von Steuer-Query-Parametern findet in der Beispielanwendung bei der Paginierung von Nachrichten-Ansichten statt, deren Umsetzung im Anhang A.4 erläutert wird.

7.6.4. Native SQL-Statements

Der `BySql`-Parameter erlaubt es, die `WHERE`-Klausel einer Abfrage in nativem SQL zu formulieren. Dabei ist die parametrisierte Form zu bevorzugen, da sie das Statement sicher gegen SQL-Injections macht (u.a. werden in Strings alle „gefährlichen“ Zeichen, wie z.B. Hochkommata, mit Escape-Zeichen versehen). Im Beispiel wird der Inhalt aller `News`-Einträge, deren Urheber die Benutzer mit den in den Variablen `id1` oder `id2` gespeicherten IDs sind, ausgegeben. Dabei müssen statt der Feldnamen die tatsächlichen Spaltennamen in der Datenbank angegeben werden. Die Fragezeichen im SQL-String werden der Reihe nach durch die übergebenen Argumente ersetzt. Da trotz der Parametrierung während der Kompilierzeit die SQL-Klausel nicht überprüft werden kann, muss dem `BySql`-Parameter eine `IHaveValidatedThisSQL`-Instanz übergeben werden, in deren Konstruktor der Name des Programmierers und das Datum übergeben werden sollen. Dies soll den gewissenhaften Umgang mit nativen SQL-Abfragen fördern, da auftretende Fehler später auf ihren Urheber zurückgeführt werden können [ChDW09], p. 93.

```

//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findallBySqlParam:
News.findAll(
  BySql(
    "betrifft_person_id_=?_or_betrifft_person_id_=?",
    IHaveValidatedThisSQL("Thomas_Fiedler", "2009-11-12"),
    id1,
    id2
  )
).foreach(msg => log.info("Inhalt:_" + msg.content))

```

Listing 7.26: Beispiel: Parametrisiertes SQL-Statement mit dem `BySql`-Parameter

Die gleiche Abfrage in unparametrierter Form, daher anfällig gegen SQL-Injection:

```

//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findallBySqlUnpar:
News.findAll(

```

```

BySql(
    "betrifft_person_id_=" + id1 + "_or_betrifft_person_id_=" + id2,
    IHaveValidatedThisSQL("Thomas_Fiedler", "2009-11-12")
)
).foreach(msg => log.info("Inhalt:_" + msg.content))

```

Listing 7.27: Beispiel: Unparametriertes SQL-Statement mit dem BySql-Parameter

Abfragen können auch vollständig in nativem SQL formuliert werden. Dazu kann die Methode `findAllByPreparedStatement` benutzt werden. Sie hat als Parameter eine anonyme Funktion, deren Parameter vom Typ `SuperConnection` ist und ein Objekt vom Typ `PreparedStatement` zurück gibt. Im Folgenden wird auf diese Weise eine einfache SQL-Abfrage generiert. Durch die Methodenaufrufe `setInt(1, 6)` und `setInt(2, 12)` werden die Parameter definiert, die an Stelle des „?“-Zeichens eingesetzt werden. Der erste `Int`-Parameter bezeichnet jeweils die ab 1 gezählte Position des zu ersetzenden „?“-Zeichens, der zweite Parameter den eingesetzten Wert. Auf diese Weise können beliebig komplizierte, gegen SQL-Injection gesicherte, Abfragen formuliert werden. Die Portierbarkeit der Anwendung wird dabei jedoch durch die Benutzung spezifischer SQL-Dialekte (in diesem Fall `H2Database`) ebenso eingeschränkt wie bei der Verwendung des `BySql`-Parameters. Das Beispiel liefert die gleiche Ausgabe wie das aus Listing 7.20.

```

//Datei src/test/scala/com/lehrkraftnews/model/NewsTest.scala, Ausschnitt findAllByPrepSql:
News.findAllByPreparedStatement({
  superconn => {
    val preparedStatement = superconn.connection.prepareStatement(
      "select *_ from nachricht where betrifft_person_id_=?_or_betrifft_person_id_="
    )
    preparedStatement.setInt(1, 6)
    preparedStatement.setInt(2, 12)
    preparedStatement
  }
}).foreach(msg => log.info("Inhalt:_" + msg.content))

```

Listing 7.28: Beispiel: Prepared SQL-Statement

In der `News`-Klasse werden auf diese Weise in der Methode `newsBySubscriber` alle Nachrichten betreffend Lehrkräfte, die ein Benutzer mit einer bestimmten `Id` abonniert hat, aus der Datenbank geholt. Die Methode findet in der Ansicht „Abonnierte“ (Nachrichten) zur paginierten Darstellung Verwendung:

```

//Datei src/main/scala/com/lehrkraftnews/model/News.scala, Ausschnitt forSubscriber:
def newsBySubscriber(id: Long, offset: Int, maxrows: Int): List[News] =
  News.findAllByPreparedStatement({ superconn => {
    val preparedStatement = superconn.connection.prepareStatement(
      "SELECT _n.*_FROM nachricht AS _n LEFT JOIN abonnement AS _a_" +
      "ON (_n.betrifft_person_id_=_a.nachrichtenquelle_person_id)" +

```

```
"WHERE_a.interessent_person_id=?_ORDER_BY_n.gueltig_bis_DESC_LIMIT_?,?"  
)  
preparedStatement.setInt(1, id.toInt)  
preparedStatement.setInt(2, offset.toInt)  
preparedStatement.setInt(3, maxrows.toInt)  
preparedStatement  
}
```

Listing 7.29: Ermitteln aller Nachrichten für einen Abonnenten

7.6.5. Zusammenfassung

Alle Query-Methoden des Traits `MetaMapper [M]` für die Modellklasse `M` akzeptieren beliebig viele Argumente, die Unterklassen von `QueryParam [M]` sind. Meist muss man wegen der Typinferenz den generischen Parameter `M` dabei nicht angeben. Dennoch werden schon durch den Compiler folgende Fehlerarten aufgedeckt, bevor das Programm ausgeführt wird.

- Abfrage von Objekten einer nicht existenten Modellklasse (z.B. wegen Tippfehler oder Umbenennungen)
- Vergleich oder Sortierung nach einem nicht existierenden Feldnamen (dito)
- Vergleich oder Sortierung nach einem Feld, das nicht zur abgefragten Modellklasse gehört
- Vergleich eines Feldes mit einem Wert falschen Typs

Damit erreichen die Datenbankabfragen mittels Mapper eine Sicherheit, die der von kompiliertem SQL gleichkommt, ohne die Schwierigkeiten einer weiteren Sprache (neben der Programmiersprache, hier Scala) einzuführen. In Java ließe sich diese Sicherheit nicht erreichen ohne einen zusätzlichen Compiler/Präprozessor.

Für spezielle Zwecke, z.B. zur Optimierung, ist es auch möglich, Mapper direkt mit SQL zu beauftragen (WHERE-Klausel oder komplette SQL-Abfrage). Dabei ist aber zu beachten, dass damit die Anwendung nicht mehr von einem Datenbanksystem zu anderen portabel ist. Außerdem muss man Vorsorge gegen SQL-Injection treffen.

7.7. Erzeugung des Datenbankschemas mit Schemifier

Lift kann auf Basis der Modellklassen ein passendes Datenbankschema erstellen bzw. es aktualisieren. Dazu steht die `schemify`-Methode des `Schemifier`-Singleton-Objekts aus dem Paket `mapper` zur Verfügung. Der Funktionsaufruf findet in der Methode `boot` der Klasse `Boot` (siehe Abschnitt 11, Seite 121) statt. Als erstes Argument erhält die Methode einen `Boolean`-Wert, der festlegt, ob die Änderungen des Datenbankschemas tatsächlich in die Datenbank geschrieben werden sollen. Das zweite Argument bestimmt die Funktion, die für das Logging der erstellten SQL-Befehle zuständig ist.²⁷ Als weitere Argumente müssen nun die von `MetaMapper` erbinden `Meta`-Objekte der Modellklassen übergeben werden, die in das OR-Mapping übernommen werden sollen.

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt schemifier:
Schemifier.schemify(true, Schemifier.infoF_, User, News, Subscription)
```

Listing 7.30: Konfiguration von `Schemifier` in `bootstrap.liftweb.Boot.boot`

7.8. CRUD-Funktionalität durch den CRUDify-Trait

Lift bietet die Möglichkeit zur automatischen Erstellung sogenannter *CRUD*-Pages. *CRUD* ist ein Akronym für *Create*, *Read*, *Update* and *Delete* (dt. Erstellen, Lesen, Aktualisieren und Löschen). Für ein Datenmodell werden dabei die Ansichten und die dazugehörige Programmlogik für diese Grundfunktionen bereitgestellt. Besonders in der frühen Entwicklungsphase und bei der Erstellung von Prototypen kann dies eine enorme Erleichterung der Arbeit und einen schnellen Einstieg in die Entwicklung bedeuten.

Die Nachrichtenverwaltung für Benutzer mit der Rolle „Administrator“ wird in der Beispielanwendung durch von Lift bereitgestellte *CRUD*-Pages realisiert.

Um *CRUD*-Pages für ein Datenmodell erstellen zu lassen, muss dessen dazugehöriges `Meta`-Objekt vom Trait `CRUDify` erben. Der `CRUDify`-Trait muss dabei mit dem Primärschlüsseltypen `Long` und der verwalteten Modellklasse `News` generisch parametrisiert werden:

```
//Datei src/main/scala/com/lehrkraftnews/model/News.scala, Ausschnitt crud:
object News extends News with LongKeyedMetaMapper[News] with CRUDify[Long, News]
```

Listing 7.31: Erzeugung von *CRUD*-Pages für die Modellklasse `News`

²⁷ Beim Start der Anwendung wird mittels der Log-Funktion `Schemifier.infoF` beim Erstellen von Datenbank-Tabellen, -Schlüsseln, -Indizes und -Constraints der entsprechende SQL-Befehl auf der Konsole protokolliert.

Die Tabelle 7.1 bietet einen Überblick über die bereitgestellten Ansichten und deren Pfade. Der CRUDify-Trait kann für diese Pfade eine Menüstruktur erstellen, deren erforderliche Einbindung in die Anwendung in Listing 11.17 auf Seite 134 beschrieben ist.

Tabelle 7.1.: Generierte Ansichten durch CRUDify

Funktion	Pfad
Listenansicht	/nachricht/list
Einzelansicht	/nachricht/view/[id]
Eintrag bearbeiten	/nachricht/edit/[id]
Eintrag erstellen	/nachricht/create
Eintrag löschen	/nachricht/delete/[id]

Abbildung 7.2 zeigt einen Ausschnitt aus der erstellten Listenansicht für die News-Klasse der Beispielanwendung. Von ihr aus lassen sich die Einträge einzeln betrachten, bearbeiten und löschen.

Nachrichten			
content	expirationdate	Lehrkraft	
Die Opel-Arbeitnehmer wollten eine Zukunft ... ehen bevor. Auch Politiker sind empört.	2009-11-28 00:00:00.0	1	View Edit Delete
Nach der Absage des Verkaufs von Opel an ... t der Autobauer bereits getilgt. mehr...	2009-11-21 00:00:00.0	1	View Edit Delete
Ein Brief, eine Podiumsdiskussion, ein T ... ge gut in ihren Reihen vorstellen. usw.	2009-11-26 00:00:00.0	1	View Edit Delete

Abbildung 7.2.: CRUDify Listenansicht

Die von CRUDify erzeugten Ansichten lassen sich durch das Überschreiben diverser Methoden anpassen. Auf der linken Seite von Abbildung 7.3 ist die unmodifizierte Bearbeitungsansicht zu sehen. Die Bezeichnungen der Eingabefelder entsprechen den Namen der Feld-Objekte aus der Modellklasse. Desweiteren lässt sich der Wert des Feldes `byUserId` zunächst nicht ändern, statt eines Eingabefeldes steht dort der Text „Can't change“.

In Listing 7.32 wird die Anzeige des Feldes `News.byUserId` konfiguriert. Der Anzeigename des Feldes `byUserId` wird durch Überschreiben der `displayName`-Methode auf „Lehrkraft“ geändert. Durch Überschreiben der Methode `validSelectValues` lassen sich mögliche Werte für eine Auswahlliste definieren. Hier werden alle Benutzer mit der Rolle `Teacher` aus der Datenbank dafür verwendet. Zu diesem Zweck wird aus einer Liste mit allen Lehrkräften eine Liste aus Tupeln erstellt, die jeweils die `id` des Benutzers und seinen Nachnamen enthalten. Diese Liste wird in einer Box zurückgegeben (anstelle des Rückgabewertes `Empty`

der überschriebenen Methode). Auf der rechten Seite von Abbildung 7.3 ist die Bearbeitungsansicht mit den in Listing 7.32 vorgenommenen Änderungen zu sehen.

```
//Datei src/main/scala/com/lehrkraftnews/model/News.scala, Ausschnitt crudModify:
class News extends LongKeyedMapper[News] with IdPK {

  /**Beschreibt Datenfeld für den Autor einer Nachricht als Fremdschlüssel für Relation zu User-Objekten*/
  object byUserId extends MappedLongForeignKey(this, User){

    /**Genutzter Spaltenname in der DB-Tabelle*/
    override def dbColumnName = "betrifft_person_id"

    /**Darstellung des Feldes auf CRUD-Seiten. Anstelle der Id wird Nachname und Vorname des Autors
     * angezeigt bzw. "k.A." für "keine Angabe", wenn es zu dieser User-Id keinen User gibt. */
    override def asHtml = Text(User.find(this).map(_fullCommaName).openOr("k.A. "))

    /**Name des Datenfeldes für CRUD-Seiten*/
    override def displayName = "Lehrkraft"

    /**Namen-Auswahlliste für CRUD-Seiten*/
    override def validSelectValues: Box[List[(Long, String)]] =
      Full(User.allTeachers.map(u => (u.id.is, u.lastName.is)))
  }
}
```

Listing 7.32: Anpassung des Datenfeldes byUserId für CRUD-Pages der Modellklasse News

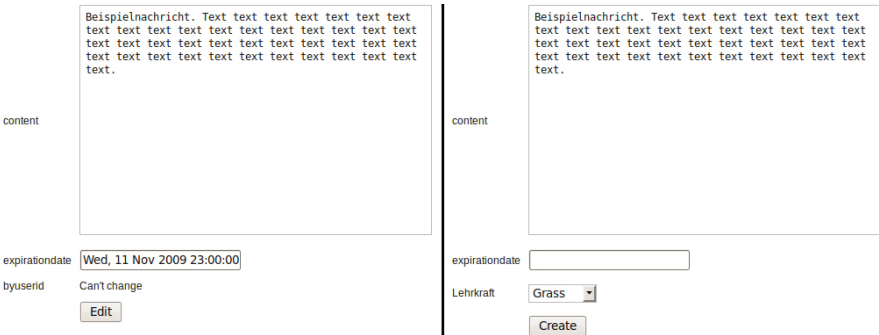


Abbildung 7.3.: CRUDify Bearbeitungsansicht

Mit der asHtml-Methode kann die Darstellung von Werten eines Feldes in CRUD-Pages definiert werden. In Abbildung 7.2 erscheinen in Spalte „Lehrkraft“ die Ids der Nachrichtenautoren. Um an dieser Stelle die Namen der Autoren aufzulisten, wurde in Listing 7.32 auch die Methode asHtml entsprechend überschrieben.

Platzierung einer CRUD-Seite

Um festzulegen, wo der generierte Inhalt im Seitenaufbau integriert wird, kann er durch Überschreiben der `pageWrapper`-Methode im Meta-Objekt mit XHTML-Elementen umhüllt werden. Der Parameter `body` repräsentiert dabei den generierten Inhalt. Im `CRUDify`-Trait wird der Inhalt standardmäßig durch ein `<lift:surround/>`-Element mit den Attributen `with="default"` `at="content"` umschlossen, welches bewirkt, dass er im `default`-Template an der Stelle `<lift:bind name="content"/>` eingefügt wird (siehe Abschnitt 8.2.2, Seite 73). Für die Klasse `News` der Beispielapplikation wird in Listing 7.33 im Meta-Objekt `News` zusätzlich festgelegt, dass das Admin-Menü und der Seitentitel angezeigt werden sollen.

```
//Datei src/main/scala/com/lehrkraftnews/model/News.scala, Ausschnitt pageWrapper:  
/**Einbettung aller News-CRUD-Seiten in das default-Template mit Admin-Menü und Titelanzeige  
 * eines "Erstellen"-Links*/  
override def pageWrapper (body: NodeSeq) = {  
  <lift:surround with="default" at="content">  
    <lift:embed what="admin_menu"/>  
    <h1><lift:Menu.title/></h1>  
  
    <div id="formBox">  
      {body}  
    </div>  
  </lift:surround>  
}
```

Listing 7.33: Einbettung der News-CRUD-Verwaltung in den Seitenaufbau

Die Darstellung der erzeugten CRUD-Pages wird durch viele weitere Methoden im `CRUDify`-Trait bestimmt. Eine genaue Beschreibung aller gebotenen Möglichkeiten kann aufgrund ihrer Vielzahl in diesem Buch nicht erfolgen. Es wird jedoch deutlich, wie vielfältig die Anpassungsmöglichkeiten sind.

7.9. ProtoUser und MegaProtoUser

Für die Erstellung von Benutzerklassen stehen in Lift die Traits `ProtoUser` und `MegaProtoUser` (und dessen Meta-Objekt `MetaMegaProtoUser`) als Bestandteil des Mapper-Frameworks zur Verfügung. Während sich der `ProtoUser`-Trait im wesentlichen darauf beschränkt, einer erbenenden Klasse Datenfelder hinzuzufügen, die für ein Standard-Benutzermodell erforderlich sind, implementiert der `MegaProtoUser` umfangreiche Funktionalitäten, die für eine Benutzerverwaltung benötigt werden. Durch das Überschreiben von Methoden können sie den Anforderungen der Webanwendung angepasst werden.

7.9.1. ProtoUser

Im folgenden Beispiel wird zunächst eine einfache Benutzerklasse `SimpleUser` unter Zuhilfenahme des `ProtoUser`-Traits implementiert.

```
import net.liftweb.mapper.{ProtoUser, KeyedMetaMapper}

object SimpleUser extends SimpleUser with KeyedMetaMapper[Long, SimpleUser]

class SimpleUser extends ProtoUser[SimpleUser]{
  def getSingleton = SimpleUser
}
```

Listing 7.34: Beispiel: Einfache Benutzerklasse mit `ProtoUser`-Trait

Der `ProtoUser`-Trait fügt der Klasse `SimpleUser` die in Tabelle 7.2 genannten Datenfelder hinzu.

Datentyp	Feldname	Beschreibung
Long	id	Benutzer-Id
String	password	Passwort
String	email	E-Mail-Adresse
String	firstName	Vorname
String	lastName	Nachname
Boolean	superUser	Einfaches Flag für Sonderprivilegien

Tabelle 7.2.: Datenfelder im `ProtoUser`-Trait

Desweiteren werden einige Hilfsmethoden, u. a. zur Formatierung des Benutzernamens, angeboten. Siehe Tabelle 7.3.

Methodenname	Ergebnis
<code>shortName</code>	Vorname und Nachname in einem String
<code>niceName</code>	Vorname, Nachname und E-Mail-Adresse in einem String
<code>niceNameWEmailLink</code>	Wie <code>niceName</code> als <code>NodeSeq</code> -Link auf die E-Mail-Adresse
<code>userIdAsString</code>	Wert des <code>id</code> -Feldes als String

Tabelle 7.3.: Hilfsmethoden im `ProtoUser`-Trait

Neben der Bestimmung des `id`-Feldes zum Primärschlüssel in der Datenbanktabelle enthält der `ProtoUser`-Trait Methoden, die nach dem Muster `fieldnameDisplayName` benannt sind.

Auf sie wird im Abschnitt „Anpassung von Funktionalitäten“ im nächsten Unterkapitel näher eingegangen.

7.9.2. MegaProtoUser und MetaMegaProtoUser

Die Benutzerverwaltung der Beispielanwendung basiert auf automatisch generierten Funktionalitäten des Mapper-Frameworks.

Die Grundlage dafür liefern die Traits `MegaProtoUser` und `MetaMegaProtoUser`. Beim `MegaProtoUser`-Trait, von dem die `User`-Modellklasse erbt, handelt es sich lediglich um eine Erweiterung des `ProtoUser`-Traits um wenige Datenfelder für die Speicherung einer `uniqueid`, der Zeitzone und der `Locale`-Variablen eines Benutzers. Der weitaus größere Zuwachs an Funktionalität erfolgt durch den `MetaMegaProtoUser`-Trait, von dem das Meta-Objekt der `User`-Klasse erbt. Das Meta-Objekt wird dadurch mit Funktionen zur automatischen Generierung von Formularen und deren Verarbeitung für wesentliche Anwendungsfälle im Bereich der Benutzerverwaltung ausgestattet.

Tabelle 7.4.: Generierte Funktionalitäten durch `MetaMegaProtoUser`

Funktion	Pfad
Registrierung mit E-Mail-Validierung	<code>user_mgt/sign_up</code>
Einloggen	<code>user_mgt/login</code>
Ausloggen	<code>user_mgt/logout</code>
Passwort-Reset mit E-Mail-Validierung	<code>user_mgt/lost_password</code>
Bearbeitung der eigenen Benutzerdaten	<code>user_mgt/edit</code>

Um die in Tabelle 7.4 beschriebenen Funktionalitäten nutzen zu können, müssen die entsprechenden Pfade unter Verwendung der Methode `LiftRules.setSiteMap` zur Pfadstruktur der Anwendung hinzugefügt werden (siehe Abschnitt 11.3 auf Seite 123).

Für die E-Mail-Validierung bei der Registrierung und beim Zurücksetzen des eigenen Passwortes wird zunächst ein zufälliger, 33-stelliger Schlüssel erzeugt und im Feld `uniqueid` in der Datenbank gespeichert. Dann wird eine E-Mail mit einem Link verschickt, dessen URL diesen Schlüssel beinhaltet. Wird der Link aufgerufen und der enthaltene Schlüssel ist in der Datenbank vorhanden, gilt die E-Mail-Adresse als validiert. Zuständig für den Versand der E-Mails ist das in Lift integrierte `Mailer`-Singleton. Dessen erforderliche Konfiguration wird im Abschnitt 11.2 (Seite 123) beschrieben.

Das Benutzerobjekt und die Benutzerklasse `User` aus der Beispielanwendung sind im Listing 7.35 dargestellt.

```

package com.lehrkraftnews.model

import _root_.com.lehrkraftnews.util.LogFactory
import net.liftweb.http.S
import xml.Text
import net.liftweb.util.FieldError
import net.liftweb.mapper.{
  Ascending, Descending, OrderBy, By,
  MetaMegaProtoUser, MegaProtoUser, MappedString
}
import net.liftweb.common.{Box, Full}

/**Meta(Kompagnion)–Objekt für die User–Klasse. Enthält instanzübergreifende Einstellungen.
 * @author Thomas Fiedler */
object User extends User with MetaMegaProtoUser[User] {
  private val log = LogFactory.create()

  /**Konstante für die Rolle Student*/
  val Student = "ST"
  /**Konstante für die Rolle Lehrkraft*/
  val Teacher = "LK"
  /**Konstante für die Rolle Administrator*/
  val Admin = "AD"

  /**Alle Rollenkürzel*/
  val allRoleValues = List(Student, Teacher, Admin)

  /**Ein nichtpersistentes User–Objekt. Dient zum Provozieren von Ausnahmen für Testzwecke,
   * z.B. in NewsSnippet.teacherSelect.*/
  val inexistent = new User
  inexistent.lastName("[Inexistent]")

  /**Name der genutzten Tabelle in der Datenbank*/
  override def dbTableName = "person"

  /**Hilfsmethode, liefert alle Benutzer mit der Rolle "Student"*/
  def allStudents= User.findAll(By(User.userRole, Student))

  /**Hilfsmethode, liefert alle Benutzer mit der Rolle Teacher*/
  def allTeachers = User.findAll(By(User.userRole, Teacher), OrderBy(User.lastName, Ascending))

  /**Liefert true, wenn der aktuelle Benutzer angemeldet ist und eine der übergebenen Rollen besitzt.
   * Die gültigen Benutzerrollen sind im Objekt User definiert. */
  def loggedInAs(roles: String*): Boolean = {
    val user = User.currentUser
    val result: Boolean = userHasOneOfRoles(user, roles: _*)
    log.debug("Request_" + S.uri + " _by_ User.currentUser="+user+" _allowed_roles_" + roles
      + ":__" + (if (result) "accepted" else "REFUSED"))
  }
  return result
}

```

```

}

/**Liefert true, wenn userBox einen Benutzer enthält, der eine der Rollen aus roles hat. */
def userHasOneOfRoles(userBox: Box[User], roles: String*): Boolean = {
  userBox match {
    case Full(user) => roles.exists(user.userRole == _)
    case _ => false
  }
}

/**Legt fest, ob E-Mail-Validation bei der Registrierung übersprungen wird*/
override def skipEmailValidation = true

/**Bestimmt, welche Felder für die Registrierung genutzt werden*/
override def signupFields = firstName :: lastName :: email :: password :: Nil

/**Bestimmt das Layout des Login-Formulars*/
//BEGIN(loginXhtml)
override def loginXhtml = {
  <form method="post" action={S.uri}>
    <table>
      <tr><td colspan="2">Login</td></tr>
      <tr><td>E-Mail</td><td><user:email/></td></tr>
      <tr><td>Passwort</td><td><user:password/></td></tr>
      <tr><td><user:submit/></td></tr>
    </table>
  </form>
}
//END(loginXhtml)

/**Bestimmt die Einbettung der generierten MetaMegaProtoUser-Seiten*/
override def screenWrap = Full{
  <lift:surround with="default" at="content">
    <h1><lift:Menu.title/></h1>
    <div id="formBox">
      <lift:bind />
    </div>
  </lift:surround>
}
}

/**Beschreibt eine Benutzer-Instanz. @author Thomas Fiedler */
class User extends MegaProtoUser[User] {

  /**Das Meta-Objekt der User-Klasse*/
  def getSingleton = User

  /**Beschreibt Datenfeld für die Rolle eines Benutzers*/
  object userRole extends MappedString(this, 2){
    override val defaultValue = getSingleton.Student
  }
}

```

```

/**Liste der durchzuführenden Validationen*/
override def validations = validateRole _ :: Nil

/**Definition der Validationsbedingung "Rolle muss legal sein"*/
def validateRole(s: String) = {
  val allowed = getSingleton.allRoleValues
  if (allowed contains s){
    List[FieldError]()
  }else{
    List(FieldError(this, Text("Feld_\\"userRole\"_\\"darf_nur_folgende_Werte_haben:_" + allowed)))
  }
}

/** Liefert alle Nachrichten eines Benutzers.
* @return Liste aller News-Objekte, die der Benutzer erstellt hat.*/
def news = News.findAll(
  By(News.byUserId, this.id), OrderBy(News.expirationDate, Descending)
)

/** Liefert alle Abonnements, die der Benutzers getätigt hat.
* @return Liste der Subscription-Objekte */
def subscriptions = Subscription.findAll(By(Subscription.subscriberId, this.id))

/** Liefert alle Benutzer, die dieser (this) Benutzer abonniert hat.
* @return Liste der User-Objekte */
def subscribedUsers: List[User] = subscriptions.flatMap(s => s.sourceId.obj)

def fullCommaName: String = this.lastName + ",_" + this.firstName
}

```

Listing 7.35: com.lehrkraftnews.model.User

Die Modellklasse User erbt vom MegaProtoUser-Trait. Nach der Zuordnung des Meta-Objekts durch getSingleton wird ein zusätzliches Datenfeld userRole definiert. Es erbt vom Trait MappedString und bildet ein String-Kürzel ab. Dieses steht in der Beispielanwendung für eine bestimmte Benutzerrolle. Definiert werden diese im Meta-Objekt der Klasse. Da die Modellklasse User durch den MegaProtoUser-Trait, den sie beerbt, mit allen für die Beispielanwendung notwendigen Datenfeldern ausgestattet ist, werden desweiteren lediglich Hilfsfunktionen definiert. Die news-Methode liefert eine Liste aller News-Objekte aus der Datenbank, die von der User-Instanz erstellt wurden, zurück. Die Methode subscriptions hat als Rückgabewert eine Liste aller Abonnements, die ein Benutzer getätigt hat, während die subscribedUsers-Methode eine Liste aller Benutzer, deren Nachrichten abonniert wurden, zurück gibt. Schließlich wird durch fullCommaName der Nachname und

der Vorname des Benutzers, durch ein Komma getrennt, als String zurückgeliefert, um später eine einheitliche Ausgabe des vollständigen Namens zu ermöglichen.

Das Meta-Objekt der User-Klasse erbt vom MetaMegaProtoUser-Trait. Die bereits erwähnten Benutzerrollen werden durch die drei Konstanten Student, Teacher und Admin definiert, denen feste String-Kürzel zugeordnet sind und die zur Abbildung der Rollen „Student“, „Lehrkraft“ und „Administrator“ dienen. Der in der Datenbank zu verwendende Tabellename wird in dbName auf „person“ festgelegt. Die Hilfsmethoden allStudents und allTeachers liefern eine Liste aller Benutzer mit der Rolle „Student“ bzw. „Lehrkraft“ aus der Datenbank zurück.

currentUser und currentUserid

Der MetaMegaProtoUser-Trait stellt die Methoden currentUser und currentUserid seinen ererbenden Klassen zur Verfügung. Sie ermöglichen den Zugriff auf das Objekt des derzeit eingeloggten Benutzers bzw. auf seine UserId (d.h. seine zur Registrierung verwendete E-Mail-Adresse). Beide Werte werden als Box geliefert. Dies ermöglicht Konstrukte der in Listing 7.36 abgedruckten Form. Dabei bildet map die Box[User] auf eine Box[String] ab, in der der Vorname des angemeldeten Benutzers enthalten sein kann. Falls kein Benutzer angemeldet ist, wird stattdessen der String "Kein Benutzer gefunden" protokolliert.

```
//Datei src/test/scala/com/lehrkraftnews/model/UserTest.scala, Ausschnitt current:  
log.info(User.currentUser.map("Vorname:" + _.firstName) openOr "Kein_Benutzer_gefunden")
```

Listing 7.36: Ausgabe des Vornamens des eingeloggten Benutzers

Anpassung von Funktionalitäten

Durch das Überschreiben von Methoden des MetaMegaProtoUser-Traits können Anpassungen der bereitgestellten Funktionalitäten durchgeführt werden. Die Methode skipEmailValidation liefert einen Wert vom Typ Boolean zurück, der bestimmt, ob bei der Registrierung eines neuen Benutzers auf eine Validierung der E-Mail-Adresse verzichtet werden soll. In der Entwicklungsphase kann es zu Testzwecken sinnvoll sein, die Validierung durch Rückgabe von false zu überspringen. Die Methode signUpFields liefert eine Liste der Felder zurück, die in den generierten Formularen für die Registrierung und die Bearbeitung von Benutzern angezeigt werden. Da die Felder email und password in jedem Fall beim Speichern einer User-Instanz validiert werden, sollten diese Felder in der Liste unbedingt enthalten sein.

Das Erscheinungsbild der generierten Formulare kann ebenfalls durch Überschreiben der dafür zuständigen Methoden bestimmt werden. Zunächst einmal zeigen wir, wie das Login-Formular im `MetaMegaProtoUser`-Trait in der Funktion `loginXhtml` aufgebaut wird (7.37).

```
def loginXhtml = { (
  <form method="post" action={S.uri}>
    <table>
      <tr><td colspan="2">{S.??("log.in")}</td></tr>
      <tr><td>{S.??("email.address")}</td><td><user:email /></td></tr>
      <tr><td>{S.??("password")}</td><td><user:password /></td></tr>
      <tr>
        <td>
          <a href={lostPasswordPath.mkString("/", "/", "")}>
            {S.??("recover.password")}
          </a>
        </td>
        <td>
          <user:submit />
        </td>
      </tr>
    </table>
  </form>
)}
```

Listing 7.37: Funktion `loginXhtml` aus dem `MetaMegaProtoUser`-Trait

Den Elementen `<user:email/>`, `<user:password/>` und `<user:submit/>` werden in der korrespondierenden Methode `login` des `MetaMegaProtoUser`-Traits die Eingabefelder für die E-Mail-Adresse und das Passwort, sowie der Submit-Button zugewiesen. (Dieser Mechanismus wird im Abschnitt 9.1 ausführlich erklärt). Der verwendete Befehl `S.??(String)` ersetzt den übergebenen String durch einen entsprechenden Eintrag in einer Properties-Datei. Im Kapitel über Internationalisierung auf Seite 157 wird auf diese Technik genauer eingegangen.

In der Beispielanwendung wurde die Methode `loginXhtml` zu Demonstrationszwecken fast vollständig aus dem `MetaMegaProtoUser`-Trait übernommen und geringfügig modifiziert. Der „Passwort vergessen“-Link wurde entfernt und die Bezeichnung der Eingabefelder geändert. Sie wurde dafür im Objekt `User` (Listing 7.35) überschrieben.

Wie beim `CRUDify`-Trait kann auch der vom `MetaMegaProtoUser`-Trait generierte Inhalt mit XHTML-Elementen „umhüllt“ werden. Dafür kann die `screenWrap`-Methode überschrieben werden. Standardmäßig ist deren Rückgabewert `Empty`, sodass eine eigene Implementierung der Methode ein Objekt vom Typ `Full` liefern muss. Das Element `<lift:bind/>` repräsentiert den generierten Inhalt. In der Beispielanwendung wird er in ein `<div>`-Element der Klasse `formBox` eingefügt, über welchem unter Verwendung des Elements

<lift:Menu.title/> eine Überschrift mit dem Titel der jeweiligen Seite eingefügt wird (siehe Abschnitt 11.3.3). Eingeschlossen werden die beschriebenen Elemente vom Element <lift:surround with="default" at="content"/>, welches bewirkt, dass sie im default-Template an der Stelle <lift:bind name="content"/> eingefügt werden. (siehe Abschnitt 8.2.2)

Beschränkungen

Wie im Abschnitt über CRUDify (siehe Listing 7.32) beschrieben, lässt sich die Methode `displayName` innerhalb einer Feld-Definition überschreiben, um den Anzeigenamen eines Feldes in generierten Formularen anzupassen. Ein `object` lässt sich jedoch nicht weiter beerben. Daher ist es nicht möglich, die Implementierungen von `displayName` in von `ProtoUser` oder `MegaProtoUser` erbedenden Benutzerklassen überschreiben zu können. Um den Anzeigenamen eines Feldes in von `ProtoUser` oder `MegaProtoUser` erbedenden Benutzerklassen dennoch ändern zu können, wurden diesen Traits Methoden, welche nach dem Muster `feldnameDisplayName` benannt sind, hinzugefügt. Die Feld-Objekt-Implementierungen belegen die Rückgabewerte ihrer jeweiligen `displayName` Methoden mit den Rückgabewerten dieser überschreibbaren Hilfsmethoden. Im Listing 7.38 ist ein Ausschnitt aus der Implementierung des `ProtoUser`-Traits dargestellt, der dies verdeutlicht:

```
def lastNameDisplayName = ??("Last_Name")

// Last Name
object lastName extends MappedString(this, 32) {
  override def displayName = fieldOwner.lastNameDisplayName
  override val fieldId = Some(Text("txtLastName"))
}
```

Listing 7.38: Definition des „Nachname“-Feldes im *ProtoUser*-Trait

Im Listing 7.39 werden die Anzeigenamen für die Felder `firstName` und `lastName` in einer eigenen, nicht in der Beispielapplikation eingesetzten, `User`-Klasse festgelegt:

```
...
class User extends MegaProtoUser[User]{

  override def firstNameDisplayName = "Vorname"
  override def lastNameDisplayName = "Nachname"
...
}
```

Listing 7.39: Festlegen des angezeigten Feldnamens in generierten Formularen

Diese Hilfsmethoden bilden jedoch die Ausnahme. So ist es zum Beispiel nicht möglich, die in der Datenbank verwendeten Spaltennamen von Feldern, die von `ProtoUser` oder `MegaProtoUser` bereitgestellt werden, zu verändern. Aus diesem Grund sind der Anpassung der beiden Traits Grenzen gesetzt. Es ist jedoch ohne großen Aufwand möglich, den Quelltext von `ProtoUser` oder `MegaProtoUser` direkt in eine eigene Benutzerklasse zu übernehmen und dort entsprechend anzupassen [GoGr09e].

Auf diese Weise lässt sich ein weiteres Problem beheben: Die Variable `editPath` ist sowohl in `MetaProtoUser` als auch in `CRUDify` vorhanden und legt dort jeweils den Pfad für die Bearbeitungsansicht fest. Somit ist es ohne die Modifikation des Quelltextes eines der beiden Traits nicht möglich, beide Pfade unabhängig voneinander zu benutzen, da man sich in der ererbenden eigenen Benutzerklasse für eine der Implementierungen entscheiden muss, indem man die Variable überschreibt:

```
object User extends User with MetaMegaProtoUser[User] with CRUDify[Long, User] {  
  override lazy val editPath = thePath(editSuffix) //MetaMegaProtoUser  
  // override lazy val editPath = Prefix ::: List(EditItem) // CRUDify  
  ...  
}
```

Listing 7.40: Auflösung der Mehrdeutigkeit beim gleichzeitigen Erben von `MetaMegaProtoUser` und `CRUDify`

8. Views und Templates

In diesem Kapitel wird die Gestaltung der Benutzungsoberfläche mit Hilfe von Views und Templates beschrieben. Ein *Template* ist dabei eine XHTML-Vorlage, in die dynamische Inhalte eingefügt werden.

8.1. View First-Pattern

In Lift ist es nicht möglich, Programmcode in Templates unterzubringen [ChDW09]. Die Programmsteuerung ist in Snippet-, View- und Comet-Actor-Klassen implementiert. Dies erleichtert die Wartung der Anwendung, schließlich können viele Fehler innerhalb von Scala-Klassen schon während des Kompilierens gefunden werden.

Ein wesentlicher Unterschied zu herkömmlichen Webframeworks stellt in Lift die Anwendung des *View-First-Patterns* dar. Dabei wird bei einem Seitenaufruf zuerst auf ein Template zugegriffen und untersucht, welche Inhalte oder Methodenaufrufe in ihm in Form von XHTML-Elementen definiert wurden. Erst dann werden die Methoden ausgeführt und Inhalte in das Template eingebunden. Diese Vorgehensweise ermöglicht einen hohen Grad der Modularisierung, da es möglich ist, vollkommen unabhängige Komponenten, z.B. verschiedene Snippets, in ein Template zu integrieren. Diese Modularisierung führt zu einer hohen Wiederverwendbarkeit von Programmfunktionen [ChDW09].

Es besteht jedoch die Möglichkeit, Präsentations-Code und Programmlogik zu vermischen. Dies ist für bestimmte Vorhaben sogar unvermeidbar, geschieht aber ausschließlich in Scala-Klassen. Die Grenze zwischen Programmsteuerungsebene und Präsentationsebene verläuft in Lift mitunter fließend [WeMa08]. Bei Lift handelt es sich ausdrücklich nicht um ein *Model-View-Controller-Framework*, an vielen Stellen wird mit diesem Entwurfsmuster gebrochen. Ein Beispiel dafür ist die umfangreiche Einmischung von Präsentations-Code in die Modell-Klassen zur Generierung von Formularen im *MetaMegaProtoUser*- und *CRUDify-Trait*.

8.2. Arbeit mit Templates

Mit den richtigen Namensraum-Deklarationen handelt es sich bei Lift-Templates um validen XHTML-Code. Zum einen wird dadurch ermöglicht, dass Templates mit bestehenden HTML-Werkzeugen bearbeitet werden können, zum anderen finden Designer keine ihnen unbekannt Syntax innerhalb der Templates vor.

8.2.1. Pfadstruktur und Benennung

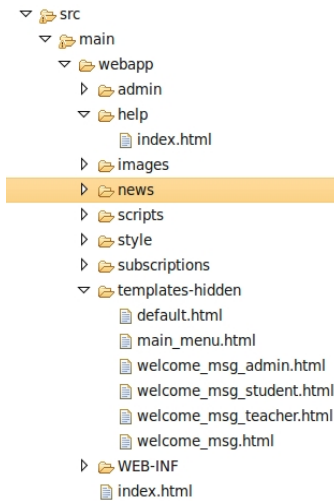


Abbildung 8.1.: Templates im *webapp*-Ordner

Abb. 8.1 zeigt einen Ausschnitt aus der Ordnerstruktur der Beispielanwendung. Templates befinden sich in dem Verzeichnis `src/main/webapp` und in dessen Unterverzeichnissen. Ausgehend vom `webapp`-Verzeichnis lassen sich Templates anhand ihrer Position in der Verzeichnisstruktur erreichen. Die URL für das Template `howto.html` in dem Ordner `src/main/webapp/help` lautet demnach `http://localhost:8080/help/howto`, die Dateieindung wird von Lift automatisch ergänzt.

Wird in der Klasse `Boot` ein `SiteMap`-Objekt definiert (siehe 11.3), ist es erforderlich, dort alle zulässigen Pfade innerhalb der Webanwendung zu registrieren, um somit den Zugriff auf

sie zu ermöglichen. Anderenfalls lässt sich nicht, wie in diesem Abschnitt beschrieben, auf Templates zugreifen.

8.2.2. surround/bind-Tag

Durch den Aufruf der URL `http://localhost:8080/help/howto` wird das Template `howto.html` aufgerufen:

```
<lift:surround with="default" at="content">
  <h1>Lehrkraftnews</h1>
  ...
  <p>
    <b>Zweck:</b> Mit dieser Applikation können Studierende sich automatisch über
    wichtige Ereignisse bezüglich ihrer Lehrkräfte benachrichtigen lassen.
  </p>
  ...
</lift:surround>
```

Listing 8.1: `src/main/webapp/help/howto.html`

Der Inhalt dieses Templates, ein Hilfetext, ist von dem Element `<lift:surround with="default" at="content">` umgeben. Dieses Element fügt die von ihm umschlossenen Elemente in ein anderes Template (bestimmt durch das `with`-Attribut) an einer durch das `at`-Attribut bestimmten Stelle ein. Im vorliegenden Fall wird der Hilfetext in das Template `default.html` an der Stelle, die durch das Element `<lift:bind name="content"/>` definiert ist, integriert. Die Dateiendung „.html“ muss dabei nicht angegeben werden, sie wird von Lift automatisch an den Wert des `with`-Attributs angehängt. Beim `default.html`-Template (Listing 8.2) handelt es sich um das Standard-Template, welches alle Einzelseiten der Beispielanwendung umrahmt. Es beinhaltet die Definition des Namespaces `lift`, einen Head-Abschnitt mit Elementen für Metadaten, die Titelangabe, CSS und JavaScript-Importe etc., sowie einen Body-Abschnitt. Durch das Einbetten von Inhalten in dieses Template wird eine einheitliche Darstellung bewirkt.

```
<!-- BEGIN(title) -->
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:lift='http://liftweb.net'>
  <head>
    <title>lehrkraftnews : <lift:Menu.title/></title>
  <!-- END(title) -->
  <meta http-equiv="content-type" content="text/html;_charset=UTF-8" />
  <meta name="description" content="" />
  <meta name="keywords" content="" />
  <lift:CSS.blueprint />
  <script id="jquery" src="/classpath/jquery.js" type="text/javascript"></script>
  <script id="json" src="/classpath/json.js" type="text/javascript"></script>
```

```

<link href="/style/lkn.css" type="text/css" rel="stylesheet" media="screen,_projection" />
<link rel="alternate" type="application/rss+xml" title="RSS" href="TestView/rss" />
</head>

<body>

<div class="container">

<div id="header">
  
  <a href="/"></a>
<div class="clear_div">
</div>

<div id="menu">
  <lift:embed what="main_menu" />
</div>

<!-- BEGIN(messages) -->
<div id="messages" class="wide_width">
  <lift:snippet type="msgs">
    <lift:error_msg>Error:</lift:error_msg>
    <lift:error_class>errorBox</lift:error_class>
    <lift:warning_msg>Warning:</lift:warning_msg>
    <lift:warning_class>warningBox</lift:warning_class>
    <lift:notice_msg>Notice:</lift:notice_msg>
    <lift:notice_class>noticeBox</lift:notice_class>
  </lift:snippet>
  <br/>
</div>
<!-- END(messages) -->

<!-- BEGIN(latestNews) -->
<div id="latest_news" class="wide_width">
  <lift:comet type="NewsReader" name="news_reader"/>
</div>
<!-- END(latestNews) -->

<div id="content" class="wide_width">
  <lift:bind name="content" />
</div>

</div>
</body>
</html>

```

Listing 8.2: src/main/webapp/templates-hidden/default.html

8.2.3. embed-Tag

Mithilfe des `lift:embed`-Elements können ebenfalls Templates in andere Templates eingebettet werden, es stellt allerdings eine Umkehr des `lift:surround`-Elements dar. Das Attribut `what` bezeichnet dabei den Namen des einzubettenden Templates. In das `default`-Template wird mithilfe von `<lift:embed what="main_menu"/>` das Template `main_menu.html` integriert, welches eine Menüführung enthält.

```
<div>
  <ul style="float:left">
    <lift:Menu.group group="main">
      <li class="menuitem"><menu:bind/></li>
    </lift:Menu.group>
  </ul>
  <ul style="float:right;">
    <li class="menuitem"><lift:HomePageSnippet.eMailIfLoggedIn/></li>
    <lift:Menu.group group="user">
      <li class="menuitem"><menu:bind/></li>
    </lift:Menu.group>
  </ul>
  <div class="clear_div"/>
</div>
```

Listing 8.3: `src/main/webapp/templates-hidden/main_menu.html`

Eine Erläuterung der Funktionsweise der `<lift:Menu>`-Elemente findet sich im Abschnitt 11.3.3.

8.2.4. ignore-Tag

Elemente, die innerhalb eines Templates von einem `<ignore>`-Element umgeben sind, werden von Lift nicht verarbeitet und auch nicht ausgegeben. Es eignet sich daher für die Deaktivierung von größeren Bereichen.

8.2.5. "templates-hidden"-Ordner

Templates, die sich im Ordner `templates-hidden` befinden, können von Besuchern der Webseite nicht direkt über ihre entsprechende URL aufgerufen werden, sie gelten als versteckt. Auf sie kann nur durch andere Templates oder Teile der Programmlogik zugegriffen werden. Dies ist nützlich, wenn Templates wie z.B. `default.html` oder `main_menu.html`

ihre Bedeutung nur im Zusammenspiel mit anderen Templates erhalten und ein alleingestellter Aufruf nicht sinnvoll ist.

Beim Einbinden von Templates mithilfe von `<surround/>` oder `<embed/>` werden diese zuerst im Ordner `webapp/templates-hidden` und dann erst im Ordner `webapp` gesucht. Aus diesem Grund kann die Angabe des Verzeichnisses `templates-hidden` im Pfad bei sich in ihm befindlichen Templates unterlassen werden.

8.2.6. Aufruf von Snippet-Methoden

Snippets werden im folgenden Kapitel ausführlich beschrieben. Vereinfacht handelt es sich bei einem Snippet um eine Methode, die innerhalb von Templates durch ein XML-Element aufgerufen werden kann. Die Kind-Elemente des Aufruf-Elements dienen als Parameter für die Methode, deren Rückgabewert vom Typ `NodeSeq` sein muss und das aufrufende Element samt seiner Kind-Elemente im Template ersetzt.

Das XML-Element für den Aufruf einer Snippet-Methode kann folgende Formen besitzen:

- `<lift:snippet Type="Klassenname:methode"/>`
- `<lift:Klassenname.methode/>`

Im Allgemeinen wird die zweite, kürzere Form verwendet. Wird der Methodename weggelassen (Form `<lift:Klassenname/>`), wird implizit die Methode `render` der Snippet-Klasse aufgerufen.

Kind-Elemente eines Snippet-Elements werden als `NodeSeq`-Parameter an die Snippet-Methode übergeben.

```
<lift:Klassenname.methode/>
  <Kind-Element><Kind-Element>...
</lift:Klassenname.methode/>
```

8.2.7. Einbettung von CometActors

Auf die Verwendung von CometActors und ihre Einbettung in Templates wird im Abschnitt 10.2 eingegangen.

8.2.8. Internationalisierung mit Templates

Neben der Vorgehensweise, die Anwendung durch das Anlegen von `.properties`-Dateien zu internationalisieren (Abschnitt 13.2, Seite 157), besteht die Möglichkeit, dies anhand vorgefertigter Templates zu tun. Zu diesem Zweck können Templates, die sich in einem gemeinsamen Verzeichnis befinden, mit einem Sprachkennungs-Suffix versehen werden. Lift öffnet in diesem Fall anhand der im Browser verwendeten Sprachkennung das passende Template. Um beispielsweise die Hilfeseite der Beispielanwendung in einer englischen Variante zur Verfügung zu stellen, erhalte das entsprechende Template den Dateinamen `howto_en.html`. Erreichbar bliebe die Hilfeseite weiterhin unter der URL `http://localhost:8080/news/howto`. Existiert für eine Sprachkennung im Browser ein passendes Template, wird dieses aufgerufen, andernfalls wird das angeforderte Template ohne Suffix geladen.

8.2.9. Head Merging

Werden Templates mit den oben genannten Methoden zusammengeführt, werden deren `<head>`-Abschnitte zusammengefügt. Dadurch ist es möglich, einem Standard-Template, welches alle Seiten umschließt, bei Bedarf weitere Elemente hinzuzufügen, die in den `<head>`-Bereich einer HTML-Datei gehören, z.B. Einbindung von CSS-Definitionen oder der Import von JavaScript-Code. In der Beispielanwendung findet dieses Feature bei der Sortierung der Nachrichtentabellen Verwendung (siehe Anhang A.5).

8.3. Arbeit mit Views

Mit Hilfe von View-Klassen lässt sich das View-First-Pattern komplett umgehen. In diesem Fall wird unter Verzicht eines Templates eine Methode innerhalb einer View-Klasse direkt aufgerufen, welche eine XHTML-Antwort zusammenbaut. Insofern ähnelt sie einem JEE-Servlet. Ihr Rückgabewert vom Typ `NodeSeq` wird anschließend wie ein Template behandelt, d.h., dass Lift-Tags wie `<embed/>`, `<surround/>` und `<bind/>` verarbeitet werden, bevor die Ausgabe im Browser erfolgt.

8.3.1. LiftView

Die Klasse `LiftView` ist die empfohlene Form, einen View zusammenzubauen. Hier ist als Beispiel die Zusammenstellung eines Feeds in der Klasse `view.Feed` dargestellt. Dieser *Feed* listet die letzten 10 eingetragenen Lehrkraftnews auf.

```

package com.lehrkraftnews.view

import net.liftweb._
import net.liftweb.common.{Box, Full, Empty, EmptyBox, Failure}
import http.{S, LiftView}
import scala.xml._
import net.liftweb.mapper._
import com.lehrkraftnews.util.{LogFactory}
import java.lang.String
import com.lehrkraftnews.model.{User, News}

object Feed {
  private val log = LogFactory.create()
}

/** View-Klasse für die Erstellung von Feeds über die eingetragenen Lehrkraftnews.
 * @author Thomas Fiedler @author Christoph Knabe */
class Feed extends LiftView {

  private val log = Feed.log

  /**Anzahl der anzuzeigenden Nachrichten*/
  val NumberOfNews = 10

  /**Verteilung gemäß URI-Pfadende an Methoden, dadurch Zugänglichmachen dieser. */
  //BEGIN(dispatch)
  override def dispatch: PartialFunction[String, () => Box[NodeSeq]] = {
    case "rss2" => _reportException(createRss2Feed)
    case "atom" => _reportException(createAtomFeed)
  }
  //END(dispatch)

  /**Holt die letzten Nachrichten aus der Datenbank, fallend geordnet nach Gültigkeitsdatum.
 * @return Liste mit News-Objekten */
  def latestNews = News.findAll(
    OrderBy(News.expirationDate, Descending), MaxRows(NumberOfNews)
  )

  /**Erstellt RSS2-Feed über die letzten Nachrichten.
 * @return XML für den Feed */
  def createRss2Feed(): NodeSeq = {
    <rss version="2.0">
      <channel>
        <title>Lehrkraftnews</title>
        <link>{S.hostAndPath}</link>
        <description>Lehrkraftnews</description>
        <language>de-de</language>
        <copyright>Hans Müller</copyright>
        {latestNews.map{ n =>
          val u = n.byUserId.obj

```

```

    val authorName = if(u.isEmpty) "k.A." else u.open_!.fullCommaName
    val nUrl = S.hostAndPath + "/news/viewSingle/" + n.id.is
    (<item>
      <title>betrifft {authorName}</title>
      <description>{n.content.is}</description>
      <link>{nUrl}</link>
      <author>{authorName}</author>
      <guid>{nUrl}</guid>
    </item>)
  })
</channel>
</rss>
}

/** Erstellt Atom-Feed über die letzten Nachrichten.
 * @return XML für den Feed */
def createAtomFeed(): NodeSeq = {
  <feed xmlns="http://www.w3.org/2005/Atom">
    <author>
      <name>Hans Müller</name>
    </author>
    <title>Lehrkraftnews</title>
    <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
    {latestNews.map{ n =>
      val u = n.userId.obj
      val authorName = if(u.isEmpty) "k.A." else u.open_!.fullCommaName
      val nUrl = S.hostAndPath + "/news/viewSingle/" + n.id.is
      (<entry>
        <title>betrifft {authorName}</title>
        <link href={nUrl}/>
        <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
        <summary>{n.content.is}</summary>
        <content>{n.content.is}</content>
      </entry>)
    }}
  </feed>
}

/** Executes the action passed and returns its result.
 * If it throws an exception, this will be logged and a Text containing a hint will be returned. */
private def _reportExeption(action: =>Box[NodeSeq]): ()=>Box[NodeSeq] = () => try {
  action
} catch {
  case e: Throwable =>
    log.warn("Failure_when_creating_feed", e)
    Full(Text("Failure_when_creating_feed. See_logfile. Cause:_" + e.toString))
}
}

```

Listing 8.4: src/main/scala/com/lehrkraftnews/view/Feed.scala

In Listing 8.4 ist die Klasse `Feed` dargestellt, in der RSS2- und Atom-Feeds für die eingetragenen Nachrichten erzeugt werden. Das Feld `hostAndPath` des `S`-Objekts (siehe Anhang A.7) erlaubt den Zugriff auf die Anfrage-URL mit Protokoll, Servernamen, Portnummer und Kontextnamen. Mithilfe der `dispatch`-Methode wird festgelegt, auf welche Methoden der Klasse über die URL zugegriffen werden kann. Nach dem Muster `/NameDerViewKlasse/caseParameter` wird die in `dispatch` ausgewählte Methode aufgerufen. Die URLs zu den Feeds lauten demnach `http://localhost:8080/Feed/rss2` und `http://localhost:8080/Feed/atom`.

Partielle `dispatch`-Funktion

Um Methoden eines Views auf diese Weise aufrufen zu können, muss die `dispatch`-Methode überschrieben und angepasst werden. Bei deren Rückgabewert handelt es sich um eine partielle Funktion, also um eine Funktion, die nur für bestimmte Argumentwerte definiert ist.

Folgender Testfall in Listing 8.5 verdeutlicht das Prinzip partieller Funktionen in Scala. Zunächst wird eine partielle Funktion `myPartial` definiert, die den Zahlwörtern "eins" und "zwei" jeweils ihre numerischen Werte zuordnet. Für alle anderen Zeichenketten ist sie undefiniert. Sodann wird sie mit den Zahlwörtern "eins", "zwei" und "drei" aufgerufen. Bei den ersten beiden wird mit `assert` geprüft, dass sie die erwarteten Zahlen liefert. Bei "drei" wird geprüft, dass sie die erwartete `MatchError`-Ausnahme wirft.

Es ist jedoch nicht nötig, eine partielle Funktion für die Werte, für die sie undefiniert ist, in die Ausnahme `scala.MatchError` hineinlaufen zu lassen. Vielmehr kann man mit der Methode `isDefinedAt` eine partielle Funktion vor ihrem Aufruf fragen, ob sie für einen bestimmten Argumentwert definiert ist. Dieses wird im letzten Drittel des abgedruckten Testfalls geprüft.

```
//Datei src/test/scala/platform/PlatformTest.scala, Ausschnitt partialFunction:
```

```
def myPartial: PartialFunction[String,Int] = {
  case "eins" => 1
  case "zwei" => 2
}

assert(myPartial("eins") === 1)
assert(myPartial("zwei") === 2)
intercept[MatchError] {
  myPartial("drei")
}

assert(myPartial.isDefinedAt("eins") === true)
assert(myPartial.isDefinedAt("zwei") === true)
```

```
assert(myPartial.isDefinedAt("drei") === false)
```

Listing 8.5: Beispiel: Partielle Funktion in Scala

Die `dispatch`-Methode (Listing 8.6) im `LiftView`-Trait liefert ein Ergebnis vom Typ `PartialFunction[String, () => Box[NodeSeq]]`

```
//Datei src/main/scala/com/lehrkraftnews/view/Feed.scala, Ausschnitt dispatch:
override def dispatch: PartialFunction[String, () => Box[NodeSeq]] = {
  case "rss2" => _reportException(createRss2Feed)
  case "atom" => _reportException(createAtomFeed)
}
```

Listing 8.6: Partielle `dispatch`-Funktion in `Feed.scala`

Dieses erwartet also ein `String`-Argument und liefert als Rückgabewert eine parameterlose Funktion, welche ihrerseits eine `Box` mit einem `NodeSeq`-Objekt zurück liefern soll. Da `Lift` Ausnahmen innerhalb einer solchen `dispatch`-Funktion verschluckt, haben wir jede verarbeitende Funktion in die private Methode `_reportException` eingeschlossen, welche jede auftretende Ausnahme protokolliert. Obwohl die Methoden `createRss2Feed` und `createAtomFeed` lediglich ein `NodeSeq`-Objekt zurück liefern, ist die `dispatch`-Methode aus Listing 8.6 gültig. Innerhalb des `LiftView`-Traits findet nämlich eine implizite Typumwandlung von `NodeSeq` nach `Box[NodeSeq]` statt.

8.3.2. InsecureLiftView

Neben dem `LiftView`-Trait, von dem die Klasse `Feed` aus der Beispielanwendung erbt, existiert der `InsecureLiftView`-Trait. Klassen, die von diesem Trait erben, kommen ohne eine `dispatch`-Methode aus; der Aufruf ihrer Methoden erfolgt implizit nach folgendem Muster:

```
/NameDerViewKlasse/methodenname
```

Die Bezeichnung „unsicherer `LiftView`“ rührt daher, dass auf diese Weise alle Methoden der Klasse aufgerufen werden können, auch solche, die nicht für die Ausgabe von Inhalten bestimmt sind. Wenn die Klasse `Feed` von `InsecureLiftView` erbte, könnte ihre Methode `latestNews` über die Angabe der URL `localhost:8080/Feed/latestNews` ausgeführt werden. Aus diesem Grund sollte bevorzugt vom `LiftView`-Trait geerbt werden, durch dessen `dispatch`-Methode der Zugriff auf Methoden explizit erlaubt werden muss.

8.4. Ausgabe von Hinweisen, Warnungen und Fehlermeldungen

Lift besitzt ein zentrales System für die Ausgabe von Hinweisen, Warnungen und Fehlermeldungen. Dabei kommt das in Lift integrierte `Msgs`-Snippet zum Einsatz. Es kann an einer geeigneten Stelle im Layout integriert werden. Durch die Zuweisung von CSS-Klassen kann die Gestaltung der Meldungen angepasst werden.

In der Beispielanwendung ist das `Msgs`-Snippet global sichtbar in den `<body>` des `default`-Templates eingebunden:

```
<!-- Datei src/main/webapp/templates-hidden/default.html, Ausschnitt messages: -->
<div id="messages" class="wide_width">
  <lift:snippet type="msgs">
    <lift:error_msg>Error:</lift:error_msg>
    <lift:error_class>errorBox</lift:error_class>
    <lift:warning_msg>Warning:</lift:warning_msg>
    <lift:warning_class>warningBox</lift:warning_class>
    <lift:notice_msg>Notice:</lift:notice_msg>
    <lift:notice_class>noticeBox</lift:notice_class>
  </lift:snippet>
  <br/>
</div>
```

Listing 8.7: Einbindung des `Msgs`-Snippets in das `default`-Template der Beispielanwendung

Die Abbildung 8.2 verdeutlicht den Zusammenhang zwischen den im Snippet-Tag gemachten Angaben und der Auswirkung auf die generierte Meldung. Es handelt sich um eine Fehlermeldung, die durch das `<lift:error_msg>`-Tag ausgegeben wird, die CSS-Klasse wird durch `<lift:error_class>` spezifiziert.

Um innerhalb von Snippet- und View-Klassen Meldungen auszugeben, stehen folgende Methoden des `S`-Objekts zur Verfügung:

- `S.notice("Hinweis")`
- `S.warning("Warnung")`
- `S.error("Fehler")`

Um jedoch eine Meldung aus einer `CometActor`-Klasse²⁸ heraus zu generieren, muss die `CometActor`-Instanz ihre Methode `notice`, `warning` oder `error` aufrufen. Dies kann man z.B. durch `super.notice("Hinweis")` bewirken.

²⁸ siehe Abschnitt 10.2

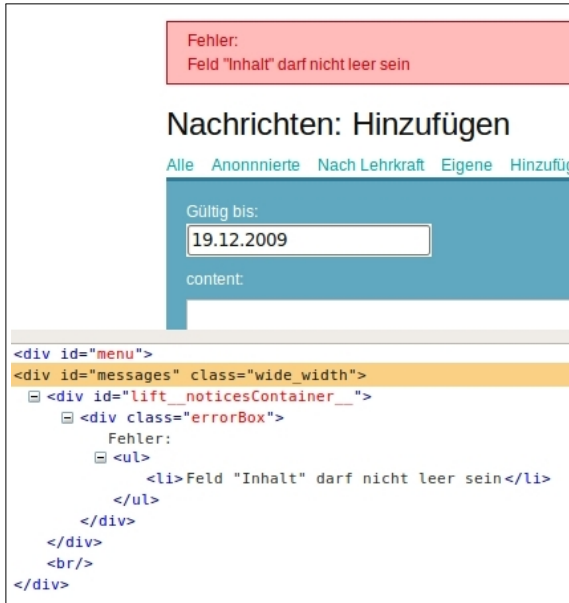


Abbildung 8.2.: Fehlermeldung und zugehöriges HTML

Um die Meldung asynchron sichtbar zu machen, bedarf es zusätzlich der Ausführung der CometActor-Methoden `partialUpdate` oder `reRender`. Listing 10.11 auf Seite 111 zeigt in einem CometActor die Erzeugung sowohl von einer Bestätigungsmeldung nach der Erstellung einer neuen Nachricht als auch von Fehlermeldungen bei fehlgeschlagener Validierung.

9. Snippets

Snippets beinhalten in Lift-Anwendungen den Programmcode für die Steuerungsebene. Zusammen mit Templates bilden sie die Basis für die Umsetzung des *View-First*-Patterns (Abschnitt 8.1). Klassen, die Snippets anbieten, müssen im Paket `snippet` abgelegt werden.

Ein *Snippet* ist eine Methode, die als Parameter ein Objekt vom Typ `NodeSeq` erhält und ein Objekt gleichen Typs zurückgibt. Ihr Aufruf erfolgt innerhalb von Templates durch ein entsprechendes Element, dessen Kind-Elemente die übergebene `NodeSeq` bilden. Nach der Ausführung der Methode wird das aufrufende Element inklusive seiner Kind-Elemente durch die von der Methode zurückgegebenen XML-Elemente (`NodeSeq`) ersetzt. Diese können wiederum Elemente aus dem Lift-Namensraum enthalten. Auf diese Weise sind Verschachtelungen möglich.

Ein einfaches Snippet ohne Parameter wird in der Beispielanwendung in der Datei `index.html` aufgerufen:

```
<lift:surround with="default" at="content">
  <lift:HomePageSnippet.welcome/>
</lift:surround>
```

Listing 9.1: `src/main/webapp/index.html`

In der mittleren Zeile befindet sich der Snippet-Aufruf. Es wird die Methode `welcome` der Klasse `com.lehrkraftnews.snippet.HomePageSnippet` aufgerufen.

Das Snippet `welcome` (Listing 9.2) gibt ein `<lift:embed/>`-Element zurück, welches ein Template mit einer Willkommensnachricht entsprechend der Rolle des derzeit eingeloggtten Benutzers einbettet.²⁹

```
//Datei src/main/scala/com/lehrkraftnews/snippet/HomePageSnippet.scala, Ausschnitt welcome:
def welcome (xhtml : NodeSeq) : NodeSeq = {
  User.currentUser.map(_UserRole.is match {
```

²⁹ Da es sich bei den Templates für die Willkommensnachricht um keine eigenständig sinnvollen Templates handelt, befinden sie sich im `templates-hidden`-Ordner

```

case User.Student => (<lift:embed what="welcome/welcome_msg_student" />)
case User.Teacher => (<lift:embed what="welcome/welcome_msg_teacher" />)
case User.Admin => (<lift:embed what="welcome/welcome_msg_admin" />)
case _ => (<lift:embed what="welcome/welcome_msg" />)
}) openOr (<lift:embed what="welcome/welcome_msg" />)
}

```

Listing 9.2: com.lehrkraftnews.snippets.HomePage

Wenn die aktuelle Rolle unbekannt ist oder kein Benutzer eingeloggt ist, wird die folgende allgemeine Willkommensnachricht eingebettet:

```

<p>
<h1>Lehrkraftnews</h1>
<a href="/user_mgt/login">Loggen Sie sich ein ...</a>
</p>

```

Listing 9.3: src/main/webapp/templates-hidden/welcome/welcome_msg.html

Diese wird wie folgt angezeigt:



Abbildung 9.1.: Eingebettete Willkommensnachricht

9.1. bind

Für die unterschiedlichen Ansichten der Nachrichten in der Beispielanwendung kommen Snippets zum Einsatz. Einem Snippet werden alle Elemente, die vom jeweiligen Snippet-Element umschlossen sind, als `NodeSeq`-Parameter übergeben. Mithilfe der `bind`-Methode werden diese XML-Elemente manipuliert und anschließend zurückgeben. Die `bind`-Methode erlaubt es, XML-Elemente durch andere XML-Elemente zu ersetzen.

Das Snippet `NewsSnippet.showSingle` für die Einzelansicht einer Nachricht wird im Template `news/viewSingle.html` wie folgt aufgerufen:

```
<!-- Datei src/main/webapp/news/viewSingle.html, Ausschnitt showSingle: -->
<lift:NewsSnippet.showSingle>
  <table>
    <tr>
      <td style="width:60px;" <b>Betrifft:</b> </td>
      <td <news:person/> </td>
    </tr>
    <tr>
      <td <b>Gültig bis:</b> </td>
      <td <news:date/> </td>
    </tr>
    <tr>
      <td style="vertical-align:top;" <b>content:</b> </td>
      <td <news:content/> </td>
    </tr>
  </table>
</lift:NewsSnippet.showSingle>
```

Listing 9.4: Auszug aus `/src/main/webapp/news/viewSingle.html`

Das Snippet-Element umschließt die komplette Tabelle. Sie wird also mit all ihren Kind-Elementen als `NodeSeq`-Parameter an die Methode `showSingle` aus der Klasse `NewsSnippet` übergeben.

9.1.1. Extraktion eines Parameters aus der URI

Durch Rewriting-Einstellungen in der Klasse `Boot` ist es in der Beispielanwendung möglich, einen Parameter aus der URI zu extrahieren, auf den dann über die Methode `S.param` zugegriffen werden kann. Ausführlich wird dies im Abschnitt 11.4 erläutert. Wird z.B. die URL `http://localhost:8080/news/viewSingle/5` aufgerufen, liefert `S.param` den Wert `Full(5)` zurück. Wird kein Parameter angegeben, ist der Rückgabewert `Empty`. Das Singleton-Objekt `BasicTypesHelpers` aus dem Paket `net.liftweb.util` besitzt die Methode

`toInt(Any)`. Sie liefert für Parameter jeden Typs einen entsprechenden `Int`-Wert zurück. Kann kein `Int`-Wert hergeleitet werden (z.B. für Objekte der Typen `Null`, `None` und `Empty` oder für `String`-Objekte, die keine Ziffern enthalten), gibt die Methode den Wert `0` zurück.

Listing 9.5 zeigt die aufgerufene Snippet-Methode. Im Snippet `showSingle` wird der Parameter `"newsId"` verwendet, um die anzuzeigende Nachricht auszuwählen. Anhand dieses Parameters wird das passende `News`-Objekt gesucht. Wird kein Objekt gefunden, wird der Text „Nicht Vorhanden“ ausgegeben, anderenfalls werden mithilfe der `bind`-Methode die Inhalte des gefundenen `News`-Objekts in das `viewSingle`-Template (Listing 9.4) eingesetzt.

9.1.2. Ersetzen von Platzhaltern durch `bind`

Der Aufruf der Methode `bind` ist der zentrale Teil des im Folgenden abgedruckten Snippets `showSingle`:

```
//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt showSingle:
def showSingle(xhtml: NodeSeq): NodeSeq = {
  val newsId = BasicTypesHelpers.toInt(S.param("newsId"))
  val msgBox = News.find(newsId)
  if(msgBox.isEmpty){return Text("Nicht_vorhanden")}
  val msg = msgBox.open_!
  bind("news", xhtml,
    "content" -> scala.xml.Unparsed(toHTML(msg.content.is)).asInstanceOf[NodeSeq],
    "date" -> Text(msg.getGermanDateString),
    "person" -> Text(msg.userId.obj.map(_.fullCommaName) openOr "nicht_verfügbar")
  )
}
```

Listing 9.5: `com.lehrkraftnews.snippet.NewsSnippet.showSingle`

Als erstes Argument erhält sie einen `String`, der das Präfix der zu ersetzenden Elemente bestimmt. Der zweite Parameter ist vom Typ `NodeSeq` und beinhaltet alle XML-Elemente, die nach zu ersetzenden Elementen durchsucht werden sollen. Im Beispiel handelt es sich um die Tabelle aus dem Template. Als weitere, optionale Parameter können `Tuple2`-Objekte der Form `String -> NodeSeq` übergeben werden.³⁰ Mit ihnen wird festgelegt, wodurch XML-Elemente mit bestimmten Namen ersetzt werden sollen. Die `bind`-Methode sorgt also zusammengefasst dafür, dass die Elemente `<news:content/>`, `<news:date/>` und `<news:person/>` jeweils durch den Inhalt der Nachricht³¹, ihr Gültigkeitsdatum bzw. den Namen

³⁰ Diese `Tuple2`-Objekte werden implizit nach `BindParam` umgewandelt.

³¹ Die private Methode `toHTML` ersetzt Zeilenumbrüche durch das `
`-Element, um das Layout des Textinhaltes beizubehalten. In Objekten des Typs `scala.xml.Unparsed` können XML-Tags übergeben werden, ohne dass diese durch Escape-Zeichen kodiert werden. `Unparsed` erbt von `NodeSeq` und kann als dieser Typ aufgefasst werden.

des Autors ersetzt werden. Alle anderen XML-Elemente bleiben unangetastet. Da der Rückgabewert der `bind`-Methode gleichzeitig der Rückgabewert des Snippets ist, wird anstelle des aufrufenden Snippet-Elements im Template die Tabelle mit den eingesetzten Werten eingefügt.

9.1.3. Iteration und bind

Um mehrere Einträge, z.B. eine Liste von Nachrichten, auszugeben, muss die `bind`-Methode mehrfach auf die gleichen Ausgangselemente angewendet werden, und das Ergebnis zu einem einzelnen `NodeSeq`-Element zusammengefügt werden.

Die Ausgabe von Nachrichten einer bestimmten Lehrkraft ist in der Beispielanwendung durch das Snippet `showByTeacher` gesteuert. Es wird im Template `_byTeacherNewsTable.html` aufgerufen und das aufrufende Element umschließt genau eine Tabellenzeile:

```
<table id="byTeacherNewsTable">
  <lift:TableSorterNewsSnippet tableId="byTeacherNewsTable"/>
  <thead>
    <tr>
      <th>Gültig bis</th>
      <th>Lehrkraft</th>
      <th>Inhalt</th>
    </tr>
  </thead>
  <tbody>
    <lift:NewsSnippet.showByTeacher>
      <tr>
        <td class="date_column"><news:date /></td>
        <td class="person_column"><news:person /></td>
        <td><news:content /></td>
      </tr>
    </lift:NewsSnippet.showByTeacher>
  </tbody>
</table>
```

Listing 9.6: `webapp/templates-hidden/tableTemplates/_byTeacherNewsTable.html`

Das `SessionVar`³²-Objekt `selectedTeacher`³³ in der Klasse `NewsSnippet` enthält ein Objekt vom Typ `Box[User]`. Ist die `Box` `Full`, wird mithilfe der `flatMap`-Methode auf jedes `News`-Objekt, das von dem entsprechenden Benutzer erstellt wurde, die `bind`-Methode angewendet. Im Gegensatz zur `map`-Methode, die in diesem Fall eine Liste mit mehreren

³² siehe Abschnitt 9.4

³³ Dessen Belegung mit einem `User` der Rolle „Lehrkraft“ über einen `AJAX`-Aufruf wird im Kapitel 10.1 beschrieben

NodeSeq-Elementen zurück liefern würde, fügt `flatMap` die Ergebnisse aus den einzelnen Anwendungen der `bind`-Methode zu einem einzelnen NodeSeq-Element zusammen. Dies ist möglich, da NodeSeq, genau wie List vom `Iterable`-Trait erbt, der die Methode `flatMap` zur Verfügung stellt. Das `showByTeacher`-Snippet gibt folglich am Ende ein NodeSeq-Element zurück, welches für jede Nachricht eine Tabellenzeile mit dem Gültigkeitsdatum, dem Namen des Autors und dem Inhalt³⁴ der Nachricht enthält. Diese Tabellenzeilen werden in die vorhandene Tabelle im Template eingefügt.

```
//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt showByTeacher:
def showByTeacher(xhtml: NodeSeq): NodeSeq = {
  val tBox = selectedTeacher.is
  if(tBox.isEmpty) return Text("Wählen_Sie_eine_Lehrkraft_aus")
  val teacher: User = tBox.open_!
  teacher.news.flatMap{ msg => {
    bind("news", xhtml,
      "date" -> Text(msg.getGermanDateString),
      "person" -> Text(teacher.fullCommaName),
      "content" -> (<a href={ "/news/viewSingle/" + msg.id.is.toString }>
        {Text(msg.content.is)}
      </a>)
    )
  }}
}
```

Listing 9.7: com.lehrkraftnews.snippet.NewsSnippet.showByTeacher

9.1.4. bind und chooseTemplate

In den bisherigen Beispielen wurde der `bind`-Methode das komplette `xhtml`-Objekt vom Typ NodeSeq übergeben, um es nach zu ersetzenden Elementen zu durchsuchen. Um innerhalb eines Snippets ein einmaliges Binding mit iterierten `bind`-Aufrufen zu kombinieren, bedarf es der Möglichkeit, die jeweiligen XML-Manipulationen auf bestimmte Elemente beschränken zu können. Zu diesem Zweck erlaubt es die Methode `chooseTemplate`, Elemente aus einem NodeSeq zu extrahieren. Als Parameter erhält die Methode das Präfix und den Namen eines Eltern-Tags sowie das NodeSeq-Objekt aus dem extrahiert werden soll. Der Rückgabewert der Methode besteht aus allen Kind-Elementen des angegebenen Eltern-Elements.

Im Snippet `showAll` sollen einmalig Links für die Paginierung³⁵ in das Template eingebunden werden. Zusätzlich soll dort für jede Nachricht eine Tabellenzeile in die Tabelle eingefügt werden.

³⁴ ausgegeben als Link zur Einzelansicht der Nachricht

³⁵ Eine genaue Beschreibung der Paginierungsfunktion erfolgt im Anhang A.4.


```

<!-- Datei src/main/webapp/news/index.html, Ausschnitt newsTable: -->
<table id="news_table">
  <lift:TableSorterNewsSnippet tableId="news_table"/>
  <thead>
    <tr>
      <th>Gültig bis</th>
      <th>Lehrkraft</th>
      <th>Inhalt</th>
    </tr>
  </thead>
  <lift:NewsSnippet.showAll>
    <tr>
      <td class="pagination" colspan="3">
        <news:pagination />
      </td>
    </tr>
    <tbody>
      <news:entries>
        <tr>
          <td class="date_column"><entry:date /></td>
          <td class="person_column"><entry:person /></td>
          <td><entry:content /></td>
        </tr>
      </news:entries>
    </tbody>
  </lift:NewsSnippet.showAll>
</table>

```

Listing 9.8: Anzeige aller News in /src/main/webapp/news/index.html

Zunächst wird die `bind`-Methode nur einmal aufgerufen. Dem Element `<news:pagination/>` wird der Rückgabewert der Methode `Paginator.paginate` zugewiesen, wobei es sich um Links für die Paginierung der Einträge handelt, welche nur einmal auf der Seite erscheinen sollen. Dem Element `<news:entries/>` wird ein durch `flatMap` zusammengesetztes `NodeSeq`-Objekt zugewiesen, welches sich aus der Anwendung von `bind` auf jedes `News`-Objekt aus `newsList` zusammensetzt. Im Gegensatz zum äußeren `bind`, welches auf das komplette `xhtml`-Objekt angewendet wird, erhält das innere `bind` nur einen Ausschnitt davon als Parameter, nämlich alle Kind-Elemente von `<news:entries>` in `xhtml`, also genau eine Tabellenzeile. Außerdem wird das Präfix der zu ersetzenden Elemente auf `entry` festgelegt.

```

//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt showAll:
def showAll(xhtml: NodeSeq): NodeSeq = {
  var totalNumberOfNews = News.count
  val newsToShowOnOnePage: List[News] = News.findAll(
    OrderBy(News.expirationDate, Descending),
    MaxRows(Paginator.maxRows),
    StartAt(Paginator.offset)
  )
}

```

```

bind("news", xhtml,
  "pagination" -> Paginator.paginate(totalNumberOfNews),
  "entries" -> newsToShowOnOnePage.flatMap( news => {
    bind("entry", chooseTemplate("news", "entries", xhtml),
      "content" ->
        (<a href={ "/news/viewSingle/" + news.id.is.toString }>{Text(news.content.is)}</a>),
      "date" -> Text(news.getGermanDateString),
      "person" -> Text(news.userId.obj.map(_fullCommaName) openOr "no_name")
    )
  })
)
}

```

Listing 9.9: com.lehrkraftnews.snippet.NewsSnippet.showAll

9.2. Formulare

Formulare können in Snippets generiert werden. Dazu muss das Snippet-Tag im Template mit einem zusätzlichen form-Parameter ausgestattet werden, dessen Wert POST oder GET betragen kann. Lift umgibt dann das vom Snippet zurückgegebene NodeSeq mit einem <form>-Element mit dem entsprechenden Request-Typ.

```

</lift:MySnippet.simpleForm form="POST">
  <form:textInput/>
  <form:submit/>
</lift:MySnippet.simpleForm>

```

Listing 9.10: Beispiel: Template für ein einfaches Formular

Für die Erstellung von Formularen stehen verschiedene Methoden des Shtml-Objekts aus dem Paket net.liftweb.http zur Verfügung, die in der Lage sind, gängige HTML-Formular-Elemente zu generieren. Mithilfe der bind-Methode können diese dann bestimmten Elementen aus dem Template zugewiesen werden.

```

def simpleForm(xhtml: NodeSeq): NodeSeq = {
  var name = "Name";

  bind("form", xhtml,
    "textInput" -> Shtml.text(name, inputString => { name = inputString },
      "class" -> "my_class"),
    "submit" -> Shtml.submit("Ok", () => Log.info("Name:_" + name))
  )
}

```

Listing 9.11: Beispiel: Einfaches Formular-Snippet

Die meisten Generator-Methoden für Formular-Elemente erhalten als Parameter einen Ausgangswert, mit dem sie belegt werden, und eine anonyme Funktion, die beim Abschicken des Formulars ausgeführt wird. Zusätzlich können dem Formular-Element mithilfe optionaler `Tuple2`-Parameter beliebig viele Attribute hinzugefügt werden.

Die `text`-Methode im Beispiel erzeugt ein Texteingabefeld, dessen Default-Wert den Wert der Variablen `name` („Name“) besitzt. In der ihr übergebenen anonymen Funktion, die als Parameter den eingegebenen Text als `String`-Objekt besitzt, wird der Wert der Variablen `name` auf den eingegebenen Wert gesetzt. Die `submit`-Methode erwartet als Parameter eine parameterlose Funktion. Im Beispiel wird bei ihrer Ausführung der Wert der Variablen `name` auf der Konsole ausgegeben.

```
<form method="post" action="/test">  
  <input class="inputfield" type="text" name="F256653433194LNU" value="Name" />  
  <input name="F2566534331951TU" type="submit" value="Ok" />  
</form>
```

Listing 9.12: Beispiel: Erzeugter HTML-Code

Der erzeugte HTML-Code macht die Funktionsweise von Lift-Formularen deutlich. Die übergebenen Wert-Parameter für das Eingabefeld und den Submit-Button werden den `<input>`-Elementen als `value`-Attribute zugewiesen. Die Formularelemente erhalten ihr entsprechendes `type`-Attribut, und der optionale `Tuple2`-Parameter setzt das `class`-Attribut des Text-`<input>`-Elements.

Als Werte für die `name`-Attribute setzt Lift GUIDs³⁶ ein, anhand derer die für die Felder definierten Funktionen im Speicher gefunden und ausgeführt werden. Bei den übergebenen Funktionen handelt es sich um Closures. Daher werden Variablen, die in ihnen referenziert, aber nicht definiert werden, in den Speicher geladen und bleiben bei der Ausführung des Closures (beim Absenden des Formulars) verwendbar, obwohl sie normalerweise nur für die Dauer der Ausführung der Snippet-Methode (beim Ausgeben des Formulars) zur Verfügung stehen würden. Aus diesem Grund wird der Inhalt des Text-Feldes, der in der Variablen `name` gespeichert ist, nach dem Abschicken des Beispielformulars auf der Konsole ausgegeben, obwohl der Kontext der Snippet-Methode dann nicht mehr existiert.

9.3. RequestVar

Standard-Snippets in Lift sind zustandlos (stateless). Das bedeutet, dass bei jedem Aufruf einer Snippet-Methode die Klasse, die es enthält, neu instanziiert wird. Im letzten Beispiel

³⁶ Globally Unique Identifier

in Listing 9.11 wurde der Variablen `name` nach dem Abschicken des Formulars der eingegebene Wert aus dem Eingabefeld zugewiesen. Zwar kann der Wert noch auf die Konsole ausgegeben werden, da das Closure des Submit-Buttons eine Referenz auf die Variable besitzt. Beim erneuten Aufruf der Snippet-Methode werden aber alle Variablen neu instanziiert, das Textfeld wird nach dem Abschicken des Formulars wieder mit dem ursprünglichen Wert der Variablen `name` („Name“) belegt.

Um Variablen während der gesamten Dauer eines Requests nutzen zu können, besteht die Möglichkeit Singleton-Objekte, die vom Trait `RequestVar` erben, zu definieren. Sie besitzen einen Typ-Parameter und können ein Objekt jeden Typs beinhalten. Aufgrund ihrer Eigenschaft als Singleton sind sie global verfügbar, daher können sie auch verwendet werden um Werte zwischen Snippets auszutauschen. Sie werden über ihren Konstruktor mit einem Default-Wert instanziiert und können über ihre `apply`-Methode mit einem Wert belegt werden. Dieser Wert bleibt für die Dauer eines Requests bestehen. Anschließend besitzt das `RequestVar`-Objekt wieder den Default-Wert. Der Zugriff auf den Wert eines `RequestVar`-Objekts erfolgt durch seine `is`-Methode.

Um zu erreichen, dass das Textfeld aus dem Beispiel nach dem Abschicken des Formulars mit dem eingegebenen Wert versehen wird, kann ein `RequestVar`-Objekt benutzt werden:

```
object name extends RequestVar[String]("Name");

def simpleForm(xhtml : NodeSeq) : NodeSeq = {

  bind("form", xhtml,
    "textInput" -> text(name.is, inputString => {name(inputString)}, "class" -> "my_class"),
    "submit" -> submit("Ok", () => {}))
}
```

Listing 9.13: Beispiel: Einfaches Formular mit `RequestVar`

Wird das Snippet zunächst aufgerufen, ohne dass das Formular abgeschickt wurde, erscheint im Textfeld der Default-Wert von `name` („Name“). Nach dem Abschicken des Formulars wird dessen Wert innerhalb des Closures durch seine `apply`-Methode auf den Eingabewert des Textfeldes gesetzt. Da der Request erst nach dem Rendern des Formulars beendet ist, wird dafür der neue Wert von `name` benutzt, im Textfeld erscheint der Eingabewert. Aktualisiert man anschließend die Seite im Browser besitzt `name` wieder seinen Default-Wert, im Textfeld erscheint wieder („Name“).

Mithilfe der `link`-Methode lassen sich `RequestVar`-Objekte komfortabel mit Werten belegen. Wie den meisten Methoden des `SHtml`-Objekts kann auch ihr eine Funktion übergeben werden. Um dies zu demonstrieren, wird das verwendete Beispiel erneut modifiziert. Zusätzlich zu dem Textfeld und dem Submit-Button wird ein Link ausgegeben, der auf das

Beispiel-Template verweist³⁷ und beim Anklicken den Wert von name auf „Neuer Name“ setzt.

```
object name extends RequestVar[String]("Name");

def simpleForm(xhtml : NodeSeq) : NodeSeq = {

  bind("form", xhtml,
    "link" -> link("beispiel", () => name("Neuer_Name"), Text("Klick")),
    "textInput" -> text(name.is, inputString => {name(inputString)}, "class" -> "my_class"),
    "submit" -> submit("Ok", () => {}))
}
```

Listing 9.14: Beispiel: Einfaches Formular mit RequestVar und link

```
<lift:NewsSnippet.simpleForm form="POST">
  <form:link/>
  <form:textInput/>
  <form:submit/>
</lift:NewsSnippet.simpleForm>
```

Listing 9.15: Beispiel: Template für Snippet in Listing 9.14

Der generierte Link hat die Form:

```
<a href="/beispiel?F1123189628608HA5=_-">Klick</a>
```

Listing 9.16: Beispiel: Generierter Link

Wieder wurde eine GUID generiert, die der Identifikation des Closures aus der link-Methode dient. In diesem Fall wird sie als GET-Parameter übergeben.

Eine sinnvollere Anwendung dieses Prinzips findet sich in der Beispielanwendung. In der Snippet-Klasse NewsSnippet ist das RequestVar-Objekt newsToDelete deklariert. Als Default-Wert wird ihm das Meta-Objekt der News-Klasse zugewiesen, dessen Feld id den Wert -1 besitzt:

```
//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt toDelete:
object newsToDelete extends RequestVar[News](News)
```

Listing 9.17: com.lehrkraftnews.snippets.NewsSnippet.newsToDelete

Im Snippet NewsSnippet.showOwn werden in einer Tabelle alle Nachrichten des eingeloggten Benutzers aufgelistet. Zu jedem Eintrag wird ein Link generiert, durch dessen Anklicken

³⁷ Es wird angenommen, dass sich das Template aus Listing 9.15 in der Datei webapp/beispiel.html befindet.

das RequestVar-Objekt `newsToDelete` mit dem jeweils angezeigten News-Objekt belegt wird (Listing 11.19 im Anhang über die Boot-Klasse). Der Link führt zum folgenden Template `delete`:

```
<!-- Datei src/main/webapp/news/delete.html, Ausschnitt main: -->
<h1>Wollen Sie die Nachricht wirklich löschen?</h1>
<div class="blue_box">
  <lift:NewsSnippet.deleteNews form="POST">
    <table>
      <tr>
        <td style="width:60px;"><b>Betrifft:</b></td>
        <td><news:person/></td>
      </tr>
      <tr>
        <td><b>Gültig bis:</b></td>
        <td><news:date/></td>
      </tr>
      <tr>
        <td style="vertical-align:top;"><b>content:</b></td>
        <td><news:content/></td>
      </tr>
      <tr>
        <td><news:cancel/></td>
        <td><news:submit/></td>
      </tr>
    </table>
  </lift:NewsSnippet.deleteNews>
</div>
```

Listing 9.18: Löschen einer Nachricht in `src/main/webapp/news/delete.html`

Dieses ruft mittels `<lift:NewsSnippet.deleteNews>` das im Folgenden abgedruckte Snippet `deleteNews` auf:

```
//Datei src/main/scala/com/lehkrftnews/snippet/NewsSnippet.scala, Ausschnitt deleteNews:
def deleteNews(xhtml: NodeSeq): NodeSeq = {
  val news = newsToDelete.is // News-Objekt aus RequestVar holen
  if (news == News){ // keine Nachricht ausgewählt
    S.redirectTo("/news/")
  }

  bind("news", xhtml,
    "content" -> scala.xml.Unparsed(toHTML(news.content.is)).asInstanceOf[NodeSeq],
    "date" -> Text(news.getGermanDateString),
    "person" -> Text(news.userId.map(u => u.fullName) openOr "no_name"),
    "submit" -> submit("Löschen", () => { doDelete(news)}),
    "cancel" -> submit("Abbrechen", () => { S.redirectTo("/news/viewOwn")})
  )
}
```

Listing 9.19: `com.lehrkraftnews.snippet.NewsSnippet.deleteNews`

Besitzt die `RequestVar`-Variable `newsToDelete` ihren Default-Wert, das `News-Meta-Objekt`, findet eine Umleitung nach `/news/` statt, da keine zu löschende Nachricht vorhanden ist. Andernfalls wird die Nachricht mithilfe der `bind`-Methode zusammen mit zwei Buttons ausgegeben, von denen der eine das Löschen der Nachricht, der andere eine Umleitung zum Template `/news/viewOwn` bewirkt.

9.4. SessionVar

`SessionVar`-Objekte verhalten sich grundsätzlich wie `RequestVar`-Objekte. Der Unterschied besteht in ihrer Lebensdauer. Während ein `RequestVar`-Objekt seinen zugewiesenen Wert nur für die Dauer eines Requests behält, ist der Wert eines `SessionVar`-Objekts für Dauer der gesamten Session verfügbar.

Im Abschnitt 10.1 wird erklärt, wie bei der Ansicht „nach Lehrkraft“ unter Benutzung eines `SessionVar`-Objekts die zuletzt ausgewählte Lehrkraft für die Dauer der Session gespeichert wird, um beim wiederholten Aufrufen der Ansicht die selben Nachrichten anzuzeigen, ohne dass eine erneute Auswahl der Lehrkraft erforderlich ist.

9.5. Stateful Snippets

Es besteht die Möglichkeit, zustandsbehaftete (stateful) Snippet-Klassen zu verwenden. Solche Klassen erben vom Trait `StatefulSnippet`. Im Gegensatz zu normalen Snippet-Klassen teilen sich nacheinanderfolgende Aufrufe ihrer Snippet-Methoden die selbe Instanz der Klasse. Somit ist es möglich, ohne die Hilfe von `RequestVar`- und `SessionVar`-Objekten für die Dauer eines Requests und länger auf eine Variable zuzugreifen, ohne dass ihr Wert durch eine Neuinstanzierung verloren geht. `StatefulSnippet`-Klassen eignen sich daher besonders für Benutzereingaben, die sich über mehrere Seiten erstrecken und in denen über Rückmeldungen ein Dialog stattfinden soll.

In der Beispielanwendung wird ein zustandsbehaftetes Snippet bei der Verwaltungsansicht für Abonnements eingesetzt. Das zugehörige Template `/subscriptions/manage` ist in Listing 9.20 abgedruckt. In dieser Ansicht können die Namen von Lehrkräften zwischen den beiden Listen „Lehrkräfte-Liste“ (Element `<formular:availableUsers/>`) und „Abonnierte Lehrkräfte“ (`<formular:subscriptionUsers/>`) verschoben werden. In der Aus-

gangssituation befinden sich alle verfügbaren Lehrkräfte in der erstgenannten Liste; die Liste „Abonnierte Lehrkräfte“ ist leer. Mithilfe des „<->“-Buttons (`<formular:move/>`) lassen sich markierte Namen von einer Liste in die andere verschieben. In der Datenbank werden die Änderungen jedoch erst durch Klicken des „Speichern“-Buttons (`<formular:save/>`) vollzogen. Entsprechende Hinweise erscheinen im Nachrichtenfenster der Anwendung.

```

<lift:surround with="default" at="content">
  <h1>Ihre Abonnements</h1>
  <div class="blue_box">
    <lift:SubscriptionSnippet.selection form="POST">
      <div class="subscription_selection">
        Lehrkräfte–Liste
        <br/>
        <formular:availableUsers />
      </div>

      <div class="subscription_selection">
        Abonnierte Lehrkräfte
        <br/>
        <formular:subscriptionUsers />
      </div>

      <div>
        <formular:move /><formular:reset /><formular:save />
      </div>

      <div class="clear_div"> </div>

    </lift:SubscriptionSnippet.selection >
  </div>
</lift:surround>

```

Listing 9.20: src/main/webapp/subscriptions/manage.html

Sobald das Template `/subscriptions/manage` aufgerufen wird, wird die im folgenden Listing 9.21 abgedruckte Snippet-Klasse `SubscriptionSnippet` initialisiert.

Deren `List subscribedUsers` wird mit den derzeit abonnierten Benutzern gefüllt, welche in der rechten Liste erscheinen sollen. Für die links darzustellende `List possibleUsers` werden von allen Lehrkräften diejenigen entfernt, die bereits abonniert sind. Handelt es sich beim Abonnenten selbst um eine Lehrkraft, wird diese ebenfalls aus der Liste entfernt.

```

//Datei src/main/scala/com/lehrkraftnews/snippet/SubscriptionSnippet.scala, Ausschnitt class:
class SubscriptionSnippet extends StatefulSnippet {

  /**Zuweisung der URL (bzw. des Ortes) an Funktion, dadurch Zugänglichmachen der Funktion */
  def dispatch = {
    case "selection" => selection _

```



```

}

/**Liste aller abonnierten Lehrkräfte */
var subscribedUsers: List[User] = User.currentUser.openOr(User).subscribedUsers

/**Liste aller Lehrkräfte abzüglich bereits abonyierter Lehrkräfte und der eigenen Person */
var possibleUsers = User.allTeachers.remove{
  e =>
    (User.currentUser.openOr(User).subscriptions.exists(_sourceId == e.id.is)) ||
    (e.id.is == User.currentUser.openOr(User).id.is)
}

/**Zustandsbehaftetes Snippet für Verwaltung von Abonnements.
 * @param xhtml Vom Snippet-Tag im Template umgebene XML-Elemente.
 * @return Durch bind modifiziertes XML für die Ansicht.
 */
def selection(xhtml: NodeSeq): NodeSeq = {

  /**Liste mit Ids von im Formular ausgewählten Lehrkräften*/
  var selectedUserIds: List[Int] = List();

  /**Verschiebt die ausgewählten Elemente in die jeweils andere Liste*/
  def updateForm(){
    selectedUserIds.foreach(a => {
      if (subscribedUsers.exists(_id.is == a)) {
        possibleUsers = subscribedUsers.filter(_id.is == a) ::: possibleUsers
        subscribedUsers = subscribedUsers.remove(_id.is == a)
      } else {
        subscribedUsers = possibleUsers.filter (_id.is == a) ::: subscribedUsers
        possibleUsers = possibleUsers.remove(_id.is == a)
      }
    })
    S.notice("Vergessen_Sie_nicht_Ihre_Änderungen_zu_speichern");
  }

  val byLastname = (u1: User, u2: User) => (u1.lastName.is compareTo u2.lastName.is) < 0
  possibleUsers = possibleUsers.sort(byLastname)
  subscribedUsers = subscribedUsers.sort(byLastname)

  /**Beschreibt Mapping von User-Objekt nach (id, Name)-Paar*/
  val userForSelectionBox = (user: User) => (user.id.toString , user.fullCommaName )

  bind("formular", xhtml,
    "availableUsers" -> SHtml.multiSelect(possibleUsers.map(userForSelectionBox), Seq(),
      u => {selectedUserIds = u.map(_toInt) ::: selectedUserIds},
      "class" -> "subscription_box"
    ),
    "subscriptionUsers" -> SHtml.multiSelect(subscribedUsers.map(userForSelectionBox), Seq(),
      u => {selectedUserIds = u.map(_toInt) ::: selectedUserIds},
      "class" -> "subscription_box"
    ),
  ),

```

```

        "move" -> submit( "<->", () => updateForm ),
        "save" -> submit( "Speichern", () => saveChanges ),
        "reset" -> submit( "Zurücksetzen", () => S.redirectTo("/subscriptions/manage") )
    )
}

/**Speichert die Änderungen*/
private def saveChanges = {
    User.currentUser.openOr(User).subscriptions.foreach(_._delete_)
    subscribedUsers.foreach{
        u =>
            Subscription.create.subscriberId(User.currentUser.openOr(User).id).sourceId(u.id).save
    }
    S.notice("Änderungen_gespeichert");
}
}

```

Listing 9.21: com.lehrkraftnews.snippet.SubscriptionSnippet

Diese beiden Listen werden, sortiert nach dem Nachnamen der Lehrkraft, in Mehrfachauswahllisten dargestellt, die an die Elemente `<formular:availableUsers/>` und `<formular:subscriptionUsers/>` aus dem Template gebunden werden. Für die Erstellung der Mehrfachauswahllisten erwartet die Methode `S.html.multiSelect` drei Parameter. Beim ersten handelt es sich um eine Liste mit `String`-Paaren, bestehend aus dem Wert (in diesem Fall der User-Id) und dem Namen des Elements in der Liste (hier der Name der Lehrkraft).³⁸ Der zweite Parameter bestimmt die vorausgewählten Elemente der Liste und besteht im vorliegenden Fall aus einer leeren Liste, da keine Vorauswahl getroffen werden soll. Der dritte Parameter ist ein Closure. Es wird beim Abschicken des Formulars auf jedes ausgewählte Element angewendet und besitzt dieses als Parameter. In der Beispielanwendung wird jedes ausgewählte Element (egal aus welcher Liste) der Liste `selectedUserIds` hinzugefügt. Schließlich wird den Listen noch ein CSS-Klassenattribut über einen optionalen Attribut-Parameter zugewiesen. Zunächst werden in der Methode `selection` nur die Listen sortiert und das Formular gerendert.

Beim Klick auf den „<->“-Button wird zusätzlich die Methode `updateForm` ausgeführt. In ihr wird für jedes in `selectedUsers` vorhandene Element geprüft, ob es sich in der linken oder in der rechten Liste befindet. Es wird aus der jeweiligen Liste entfernt und der anderen hinzugefügt. Mit der `S.notice`-Methode wird der Hinweis ausgegeben, dass die Änderungen noch nicht gespeichert sind.

Wird der „Ok“-Button geklickt, wird die Methode `saveChanges` aufgerufen, in der zunächst alle Abonnements des aktuellen Benutzers (`User.currentUser`) aus der Datenbank ge-

³⁸ Die Abbildung eines Benutzers auf ein solches Paar wird im Beispiel als Funktion `userForSelectionBox` gespeichert und in der `bind`-Methode bei beiden Listen verwendet.

löscht werden. Anschließend wird für jedes in `subscribedUsers` befindliche `User`-Objekt ein neues Abonnement erstellt, und die Speicherung der Änderungen bekannt gegeben.

Beim Betrachten des vom *Stateful*-Snippet generierten, nachfolgenden HTML-Formulars fällt auf, dass in Erweiterung des Templates `/subscriptions/manage` aus Listing 9.20 ein `input`-Element vom Typ `hidden` in das Formular eingefügt wurde. Anhand seines `name`-Attributs, einer GUID, wird sichergestellt, dass für die Verarbeitung des Formulars die selbe Snippet-Instanz benutzt wird wie die, die es erstellt hat.

```
<form method="post" action="/subscriptions/manage">

  <input name="F1134553887556DKM" type="hidden" value="true" />

  <div class="subscription_selection">
    Lehrkräfte–Liste <br />
    <select name="F1134553887557WI4" multiple="true" class="subscription_box">
      <option value="9">Müller, Karl</option></select>
  </div>

  <div class="subscription_selection">
    Abonnierte Lehrkräfte <br />
    <select name="F1134553887558EN1" multiple="true" class="subscription_box">
      <option value="7">Orf, Karl</option>
      <option value="6">Wampe, Werner</option>
      <option value="8">Wonka, Willi</option></select>
  </div>

  <div>
    <input name="F1134553887559S3B" type="submit" value="&lt;– &gt;" />
    <input name="F1110299073578TBT" type="submit" value="Zurücksetzen" />
    <input name="F1134553887560ZEL" type="submit" value="Speichern" />
  </div>

  <div class="clear_div">
</form>
```

Listing 9.22: Generiertes Formular (mit Beispieldaten) aus der Abonnement-Verwaltung

Navigiert der Benutzer von der Abonnement-Ansicht weg zu einer anderen Ansicht, und anschließend wieder zurück, wird eine neue Instanz der Snippet-Klasse erstellt, da in diesem Fall keine Referenz in Form einer GUID vorhanden ist. Nicht gespeicherte Änderungen an den Abonnements gehen deshalb verloren. Auf diese Weise funktioniert der „Zurücksetzen“-Button. Er löst explizit die `redirectTo`-Methode des `S`-Objekts aus, mit deren Hilfe Umleitungen auf URLs realisiert werden können. *Stateful*-Snippets besitzen eine eigene Variante von `redirectTo`, die bewirkt, dass für Snippet-Methodenaufrufe im Ziel-Template die gleiche Instanz des Snippets verwendet wird. Würde also statt `S.redirectTo` nur `redirectTo`

verwendet werden, würden die Änderungen nicht zurückgesetzt werden, da in diesem Fall die selbe Instanz der Snippetklasse genutzt würde.

10. AJAX und Comet

AJAX und Comet stellen Ansätze dar, Webanwendungen mit einem höheren Maß an Interaktivität zu versehen als es durch den traditionellen Request/Response-Zyklus möglich ist. Klassische Webanwendungen laden nach jeder Benutzerinteraktion eine komplett neue Seite. AJAX und Comet erlauben es, nur die Teile einer Webseite nachzuladen, die von einer Interaktion (AJAX) oder einer Änderung auf der Server-Seite (Comet) betroffen sind [PhDi09].

Lift stellt für beide Ansätze Klassen und Methoden bereit, durch die die Verwendung der Technologien abstrahiert und dadurch vereinfacht wird. Die Implementierung des notwendigen JavaScript-Codes wird dabei größtenteils vom Framework übernommen.

10.1. AJAX

AJAX (Asynchronous JavaScript and XML) bezeichnet den Ansatz, Web-Inhalte auf Benutzeranfrage asynchron zu übermitteln. Dazu steht das Objekt `XMLHttpRequest` zur Verfügung, auf das im Browser mit der Scriptsprache JavaScript zugegriffen werden kann, um HTTP-Requests zu versenden und die erhaltene Antwort zu verarbeiten. Dies geschieht gewöhnlich ebenfalls durch JavaScript-Methoden, die in Lage sind, das DOM-Modell der Webseite Client-seitig zu manipulieren und somit den empfangenen Inhalt in die Webseite zu integrieren.

In der Ansicht „Nachrichten: nach Lehrkraft“ in der Beispielanwendung werden alle Nachrichten, die von einer ausgewählten Lehrkraft erstellt wurden, aufgelistet. Der Bediener wählt unter *Betreffend die Lehrkraft*: eine aus und klickt dann den *Abschicken*-Button. Die Reaktion darauf, nämlich die Ersetzung der angezeigten Nachrichtentabelle, wird mit einem AJAX-Aufruf realisiert.

Die Tabelle, in der die Nachrichten erscheinen sollen, ist in ein eigenes Template ausgelagert und wird mittels `<lift:embed>` innerhalb des Div-Elements mit der Id `byTeacherNewsTableInsertDiv` eingebunden. Innerhalb dieses ausgelagerten Templates `_byTeacherNewsTable`, Listing 10.2 befindet sich der Aufruf des Snippets `showByTeacher`, dessen Funktionsweise im Abschnitt 9.1.3 erläutert wurde.

```

<!-- Datei src/main/webapp/news/viewByTeacher.html, Ausschnitt selectShow: -->
<lift:NewsSnippet.teacherSelect form="POST">
  Betreffend die Lehrkraft:
  <teacherSelect:select />
  <teacherSelect:submit />
</lift:NewsSnippet.teacherSelect>

<div id="byTeacherNewsTableInsertDiv">
  <lift:embed what="tableTemplates/_byTeacherNewsTable"/>
</div>

```

Listing 10.1: Auszug aus src/main/webapp/news/viewByTeacher.html

```

<table id="byTeacherNewsTable">
  <lift:TableSorterNewsSnippet tableId="byTeacherNewsTable">
  <thead>
    <tr>
      <th>Gültig bis</th>
      <th>Lehrkraft</th>
      <th>Inhalt</th>
    </tr>
  </thead>
  <tbody>
    <lift:NewsSnippet.showByTeacher>
    <tr>
      <td class="date_column"><news:date /></td>
      <td class="person_column"><news:person /></td>
      <td><news:content /></td>
    </tr>
    </lift:NewsSnippet.showByTeacher>
  </tbody>
</table>

```

Listing 10.2: webapp/templates-hidden/tableTemplates/_byTeacherNewsTable.html

Die im Template (Listing 10.1) aufgerufene Snippet-Methode `NewsSnippet.teacherSelect` fügt ein Formular ein, welches aus einer Auswahlliste mit allen Lehrkräften und einem Submit-Button besteht. Dabei sorgt die Methode `SHtml.ajaxForm` dafür, dass das generierte Formular als AJAX-Formular ausgegeben wird. Statt eines herkömmlichen Submits wird beim Absenden eine JavaScript-Funktion aufgerufen, die die Verarbeitung des Formulars übernimmt.

```

//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt teacherSelect:
def teacherSelect(xhtml: NodeSeq): NodeSeq = {

  /**Liste aus (id, Name)–Tupeln aller Lehrkräfte für die Auswahlliste.
   * Der erste Eintrag ist eine nicht–existente Lehrkraft zum testweisen Provozieren einer Ausnahme.*/

```

```

val teachers0 = User.inexistent :: User.allTeachers
val teachers = teachers0.map(teacher => (teacher.id.toString, teacher.fullCommaName))

/**Wird beim Absenden des Formulars ausgeführt */
val onSelectAction = {
  id: String =>
  val teacherBox = User.find(id)
  if(teacherBox.isEmpty){
    val exc = new Exc("Lehrkraft_mit_Id_{0}_nicht_gefunden.", id)
    NewsSnippet.log.debug("Throwing_" + exc)
    throw exc
  }
  selectedTeacher(teacherBox)
  CmdPair(
    SetHtml("byTeacherNewsTableInsertDiv",
      (<lift:embed what="tableTemplates/_byTeacherNewsTable" />)),
    JsRaw("$$$(<document>).updateSorter('byTeacherNewsTable');_$$$")
  )
}

SHtml.ajaxForm{
  bind("teacherSelect", xhtml,
    "select" -> select(teachers, Empty, onSelectAction),
    "submit" -> submit("Abschicken", () => {})
  )
}
} //teacherSelect

```

Listing 10.3: com.lehrkraftnews.snippet.NewsSnippet.teacherSelect

Zu beachten ist, dass Closures für Submit-Buttons beim Absenden von AJAX-Formularen nicht ausgeführt werden. Wird die Ausführung eines zusätzlichen Closures der Form `() => {}` benötigt, kann ein `hidden`-Element an das Formular angefügt werden:

```

SHtml.ajaxForm{
  bind("teacherSelect", xhtml,
    "select" -> select(users, Empty, onSelectAction),
    "submit" -> submit("Abschicken", () => { /*wird nie ausgeführt*/ }) ++
    hidden(() => { Log.info("Ausgabe") })
  )
}

```

Listing 10.4: Beispiel: Zusätzliches Closure durch hidden-Element in AJAX-Formularen

Mit der Auslagerung des Closures für die Auswahlliste in den Wert `onSelectAction` wird versucht, eine größtmögliche Lesbarkeit des Programmcodes zu erreichen. Sein Parameter ist die Id der ausgewählten Lehrkraft als String. Falls zu dieser Id keine Lehrkraft gefunden wird, wird eine Ausnahme ausgelöst. Dies wird benötigt, um korrektes Melden von

Ausnahmen testen zu können. Näheres dazu im Abschnitt 12.3. Sodann wird im Closure das `SessionVar`-Objekt `selectedTeacher` (siehe Abschnitt 9.4) mit dem der Id entsprechenden `User`-Objekt belegt. Die Instanziierung des `SetHtml`-Objekts mit der Id des `Div`-Elements im `Template` und einem `NodeSeq`-Parameter bewirkt, dass beim Abschicken des Formulars der Inhalt des `Div`-Elements durch den `NodeSeq`-Parameter ersetzt wird. Den dafür benötigten JavaScript-Programmcode bindet `Lift` in die generierte Seite ein. Das `JsRaw`-Objekt erlaubt es, nativen JavaScript-Code in die Seite einzufügen, der beim Abschicken des Formulars auszuführen ist. Bei dem übergebenen Code handelt es sich um die notwendige Neuinitialisierung der `jQuery`-Tabellensortierung³⁹. Da `Lift` den JavaScript-Code jedoch nur für das letzte instanziierte Objekt dieser Art generiert, werden beide Objekte durch das `CmdPair`-Objekt zusammengefasst. Der generierte Code wird dadurch nacheinander ausgeführt.

Durch das erneute Einbinden des ausgelagerten Templates `_byTeacherNewsTable` durch das `SetHtml`-Objekt wird der darin enthaltene Aufruf des Snippets `showByTeacher` erneut ausgeführt. Das `SessionVar`-Objekt kann durch das AJAX-Formular asynchron belegt werden, wodurch, ebenfalls asynchron, die entsprechenden Nachrichten durch das erneut aufgerufene Snippet ausgegeben werden. Die demonstrierte Möglichkeit der Verschachtelung von Snippet-Aufrufen mithilfe des `<lift:embed>`-Tags bietet bei der Erstellung von AJAX-Anfragen ein hohes Maß an Flexibilität.

Neben dem `SetHtml`-Objekt stehen weitere Objekte für die Generierung von JavaScript-Code zur Verfügung. Die wichtigsten sind `AppendHtml` und `PrependHtml`. Sie haben die gleichen Parameter wie `SetHtml`. Statt jedoch den Inhalt des durch den ersten Parameter bestimmten Elements durch die übergebene `NodeSeq` zu ersetzen, wird er bei diesen beiden Methoden vor bzw. hinter den bestehenden Inhalt eingefügt.

Für die Dauer einer AJAX-Anfrage kann eine Ladeanzeige eingeblendet werden; erläutert wird dies im Abschnitt 11.1.

10.2. Comet

AJAX setzt als Auslöser für die Übertragung von Inhalten eine Anfrage des Benutzers voraus. Der Ansatz, Inhalte vom Server an den Client als Reaktion auf eine serverseitige Veränderung zu versenden, wird unter dem Begriff *Comet* zusammengefasst.

`Lift` nutzt für die Umsetzung von Comet-Verbindungen `XMLHttpRequest-Long-Polling`. Es wird das gleiche Prinzip wie bei AJAX-Anfragen verwendet, die Antwort des Servers erfolgt

³⁹ Siehe Anhang: Tablesorter

jedoch erst, wenn Inhalte vorliegen, die an den Client gesendet werden sollen. In der Zwischenzeit bleibt die Verbindung bestehen. Sobald die Inhalte vom Client empfangen wurden, sendet dieser eine neue Anfrage, um wieder eine Verbindung aufzubauen. Viele Browser limitieren die Anzahl der simultanen Verbindungen zu einem Server auf maximal zwei Verbindungen. Um mit dieser Einschränkung zurechtzukommen, verwendet Lift die folgende Strategie. Wenn eine Long-Polling-Verbindung zu einem Server besteht, und Lift einen weiteren Request der selben Session bemerkt, beendet Lift die Long-Polling-Verbindung und geht zum konventionellen Polling, also dem schnellen, wiederholten Absenden einzelner AJAX-Requests an den Server, über. [GoGr09b]

Das Comet-Prinzip wird in Lift mithilfe von CometActor-Klassen umgesetzt. Diese befinden sich im Unterpaket comet der Applikation und erben vom Trait `net.liftweb.http.CometActor`, der wiederum vom Trait `net.liftweb.actor.LiftActor` erbt. Lift-Actors implementieren nebenläufige Prozesse, mit denen durch den Austausch von Messages kommuniziert werden kann (siehe Anhang A.6).⁴⁰

10.2.1. Autonome CometActors

Info:
E-Mails werden versendet!

Nachrichten: Hinzufügen

Anonnierte [Alle](#) [Nach Lehrkraft](#) [Eigene](#) [Hinzufügen](#)

Gültig bis:

content:

Noch 1 E-Mails zu versenden...

Versand erfolgreich: [tliedler@th-berlin.de](#)
[s16256@th-berlin.de](#)
[thomas@jtz.de](#)

Versand Fehlgeschlagen: [adresseohneATzeichen.de](#)

Nachricht als E-Mail versenden?

Abbildung 10.1: Ansicht: Nachricht hinzufügen

⁴⁰ Ab Lift 2.0 werden nicht mehr die Standard-Scala-Actors, sondern Lift-eigene Actors durch Lift verwendet. Auf den *ScalaDays2010* wurde dies damit begründet, dass sich die Scala-Actors in bestimmten Situationen inperformant verhielten.

Abbildung 10.1 zeigt die Ansicht „Nachricht hinzufügen“ bzw. „Nachricht bearbeiten“ während des Versands einer Nachricht als E-Mail. Das dazugehörige Template ist `src/main/webapp/news/edit.html`. Der große blau hinterlegte Bereich enthält die ansichtsspezifischen Teile. Dessen linke Hälfte ist ein Comet-Formular mit den Eingabefeldern für die Nachricht. Der entsprechende Template-Ausschnitt ist in Listing 10.5 dargestellt.

```

<!-- Datei src/main/webapp/news/edit.html, Ausschnitt formular: -->
<div id="mailcomet_formular">
  <lift:comet type="MailComet" name="edit" form="POST">
    <table id="form_table">
      <tr>
        <td> Gültig bis:<br/> <form:date /> </td>
      </tr>
      <tr>
        <td colspan="2"> content:<br/> <form:content/> </td>
      </tr>
      <tr>
        <td> Nachricht als E-Mail versenden? <form:sendCheck/> </td>
        <td> <form:save/> </td>
      </tr>
    </table>
  </lift:comet>
</div>

```

Listing 10.5: Comet-Formular in `src/main/webapp/news/edit.html`

Die CometActor-Klasse `MailComet` wird hier durch das `<lift:comet>`-Tag instanziiert, wobei das `type`-Attribut den Namen der Klasse bestimmt. Das `name`-Attribut kann frei gewählt werden und wird zur Unterscheidung verschiedener Instanzen einer CometActor-Klasse benötigt. CometActors sind zustandbehaftet. Nach der erstmaligen Instanziierung wird bei gleichem `name`-Attribut immer die selbe Instanz, auch nach einem erneuten Laden der Seite, verwendet. Da der CometActor für die Erstellung von Nachrichten ein Formular ausgeben soll, wird zusätzlich das Attribut `form` mit dem Wert `POST` benötigt.

Die rechte Hälfte des blauen Bereichs aus Abbildung 10.1 enthält die asynchronen Rückmeldungen über den Mail-Versand. Oben sieht man eine grüne Statuszeile „Noch n E-Mails zu versenden...“. Deren Anzahl n wird fortlaufend aktualisiert. In der Spalte „Versand erfolgreich“ werden fortlaufend die E-Mail-Adressen protokolliert, an die erfolgreich versandt wurde. In der rechten Spalte „Versand fehlgeschlagen“ werden fortlaufend die E-Mail-Adressen protokolliert, bei denen ein adressbezogener Versandfehler (`SendFailedException`) aufgetreten ist. Der den besprochenen Rückmeldungsbereich generierende Template-Ausschnitt ist in Listing 10.6 dargestellt.

```

<!-- Datei src/main/webapp/news/edit.html, Ausschnitt status: -->
<div id="addresses-div" style="float:right;_width:470px;">
  <table>

```

```

<tr>
  <td colspan="2" id="statusMessage" style="color:green;">
    <lift:ignore> SUCCESSMESSAGE IS INSERTED HERE</lift:ignore>
  </td>
</tr>
<tr>
  <td style="vertical-align:top;">
    Versand erfolgreich: <br/>
  </td>
  <td style="vertical-align:top;">
    Versand Fehlgeschlagen:
  </td>
</tr>
<tr>
  <td id="addressesSuccess" style="vertical-align:top;">
    <lift:ignore> ADDRESSES ARE INSERTED HERE</lift:ignore>
  </td>
  <td id="addressesFail" style="color:_red;_vertical-align:top;">
    <lift:ignore> ADDRESSES ARE INSERTED HERE</lift:ignore>
  </td>
</tr>
</table>
</div>

```

Listing 10.6: Rückmeldungsbereich in `src/main/webapp/news/edit.html`

Die relevanten Elemente im Rückmeldungsbereich des Templates sind die Tabellenzellen mit den Ids `statusMessage`, `addressesSuccess` und `addressesFail`. In sie werden beim Versenden der Nachricht an die einzelnen Abonnenten die jeweilige Statusmeldung, sowie die E-Mail-Adressen, an die die Nachricht versendet bzw. nicht versendet werden konnte, eingefügt.

Die Klasse MailComet

Die Ansichten „Nachricht hinzufügen“ und „Nachricht bearbeiten“ teilen sich das Template und den CometActor `MailComet`. Soll eine Nachricht hinzugefügt werden, wird das `SessionVar`-Objekt `selectedNewsVar` (Listing 10.7) mit einer neu erstellten Nachricht des aktuellen Benutzers belegt (siehe dazu Abschnitt 11.3.4). Soll eine bestehende Nachricht bearbeitet werden, wird `selectedNewsVar` mithilfe eines Closures im Snippet `NewsSnippet.viewOwn` innerhalb einer Link-Generatormethode mit dem entsprechenden `News`-Objekt belegt.

```

//Datei src/main/scala/com/lehrkraftnews/comet/MailComet.scala, Ausschnitt selectedNewsVar:
object selectedNewsVar extends SessionVar[News](
  News.create.byUserId(User.currentUser.openOr(User).id)

```

)

Listing 10.7: MailComet: Definition des SessionVar-Objekts

Bei der Initialisierung von MailComet werden die Variablen `sendEmails` und `emailsToSend` (Listing 10.8) auf `false` bzw. 0 gesetzt.

```
//Datei src/main/scala/com/lehrkraftnews/comet/MailComet.scala, Ausschnitt hilfsvariablen:
/**Soll die Nachricht als E-Mail versendet werden?*/
var sendEmails = false

/**Zähler für noch zu versendende E-Mails*/
var emailsToSend: Int = 0
```

Listing 10.8: MailComet: Hilfsvariablen

Sie werden eingesetzt, um zu entscheiden, ob E-Mails verschickt werden (wählbar in Ansicht Abbildung 10.1) und falls ja, um zu zählen, an wie viele Abonnenten noch zu versenden ist. Der letztgenannte Wert wird nach dem Versand jeder einzelnen E-Mail aktualisiert und in einer ansichtspezifischen Statuszeile ausgegeben.

Die render-Methode von MailComet

CometActor-Klassen besitzen eine `render`-Methode (Listing 10.9), die nach der Initialisierung der Klasse ausgeführt wird:

```
//Datei src/main/scala/com/lehrkraftnews/comet/MailComet.scala, Ausschnitt render:
override def render = SHtml.ajaxForm{
  bind("form",
    "date" -> SHtml.text(
      selectedNews Var.getGermanDateString,
      s => selectedNews Var.expirationDate(stringToDate(s)),
      "id" -> "entrydate"
    ),
    "content" -> SHtml.textarea(selectedNews Var.is.content.is, selectedNews Var.content(_)),
    "sendCheck" -> SHtml.checkbox(true, sendEmails = _),
    "save" -> SHtml.submit("Speichern", ()=>{} )
  ) ++ SHtml.hidden(doSave)
}
```

Listing 10.9: MailComet: render-Methode

Ihr Rückgabewert ersetzt im Template das `<lift:comet>`-Tag und seine Kindelemente. Die `render`-Methode ist im Gegensatz zu Snippet-Methoden parameterlos. Aufgrund des da-

durch fehlenden NodeSeq-Parameters besitzen CometActors eine eigene bind-Methode, die als Parameter nur das Präfix der gebundenen Elemente und die BindParam-Objekte übergeben bekommt. Sie arbeitet implizit auf den Kind-Elementen des Comet-Tags aus dem Template. Die Kindelemente des Comet-Tags stehen innerhalb der Klasse als Variable defaultXml zur Verfügung. Somit wäre es theoretisch auch möglich, die bind-Methode des Helpers-Objektes zu benutzen, die auch in Snippets verwendet wird. Jedoch müssten dafür die übergebenen BindParam-Objekte explizit nach Helpers.BindParam gecastet werden:

```
Helpers.bind("form", defaultXml,
  ("element" -> Text("Beispieltext")).asInstanceOf[Helpers.BindParam]
)
```

Listing 10.10: Verwendung der Helpers.bind-Methode innerhalb von CometActor-Klassen

Im Closure des Datum-Feldes wird das Gültig-bis-Datum gesetzt. Die Umwandlung des eingegebenen Strings in ein Date ist in eine private Methode stringToDate ausgelagert. Wenn der String nicht als deutsches Kalenderdatum interpretiert werden kann, wird das Datumfeld mit null belegt, was bei der Validierung des News-Objektes zu einem Fehler führt (siehe Abschnitt 7.4.6). Durch eine Checkbox kann der Benutzer entscheiden, ob die Nachricht durch E-Mails versendet wird. Beim Abschicken des Formulars wird durch Shtml.hidden(doSave) die folgende Methode doSave ausgeführt:

Die doSave-Methode von MailComet

```
//Datei src/main/scala/com/lehrkraftnews/comet/MailComet.scala, Ausschnitt doSave:
def doSave(): Unit = {
  val news = selectedNewsVar.is
  news.validate match {
    case Nil => {
      if (news.content=="Unsinn") throw new IllegalArgumentException("Unsinn_verboten")
      news.save
      val txt = if (sendEmails) {
        sendNewsMail
        "E-Mails_werden_versendet!"
      } else "Nachricht_gespeichert"
      super.notice(txt)
      NewsMaster ! NewPost(news)
      this ! ResetSelectedNews
    }
    case errors: List[FieldError] => {
      errors.foreach(e => super.error(e.msg))
    }
  }
  //match
  partialUpdate(Noop);
}
```

Listing 10.11: MailComet: doSave-Methode

In `doSave` wird zunächst das erstellte `News`-Objekt aus `selectedNewsVar` mithilfe seiner `validate`-Methode validiert. Ist die Validierung nicht erfolgreich und die Methode liefert eine Liste mit `FieldError`-Objekten zurück, wird für jedes Element der Liste eine Error-Nachricht ausgegeben. Statt der Methode `S.error`, die für diese Zwecke in Snippets genutzt werden kann, muss in einem `CometActor` dessen Methode `error` verwendet werden. Diese sendet eine Message an die eigene Instanz, die aus einem `Error`-Objekt (im diesem Fall mit dem Fehlertext des `FieldError`-Objekts) besteht. Die Methode `partialUpdate` sorgt dafür, dass die Error-Nachrichten im dafür vorgesehenen Bereich im `default`-Template ausgegeben werden. Als Parameter erhält sie ein Objekt vom Typ `JsCmd`. Mit Objekten dieser Art kann JavaScript-Code in die ausgegebene Seite eingefügt und anschließend ausgeführt werden. Das `Noop`-Objekt (`Noop` = No Operation) kann übergeben werden, wenn kein zusätzlicher Code eingefügt werden soll. Die Ausgabe der Meldung findet trotzdem statt.

Bei erfolgreicher Validierung des `News`-Objekts ist die `FieldError`-Liste leer (`Nil`). Die erste Anweisung mit Prüfung vom `news.content` auf "Unsinn" ist nützlich für den Test des Ausnahmemeldens (siehe Abschnitt 12.4). Dann wird das Objekt gespeichert und falls der Benutzer dies gewählt hat, als E-Mails versendet. Eine entsprechende Notiz wird sofort mittels der `CometActor`-Methode `notice` ins `default`-Template ausgegeben. Den Versand übernimmt die private Methode `sendNewsMail` in Listing 10.12.

Bei dem Objekt `NewsMaster` handelt es sich um einen `LiftActor`, der die `NewsReaderCometActors` verwaltet, die für die Anzeige neu erstellter Nachrichten zuständig sind. (Sie werden im Anschluss erläutert) Diesem Actor-Objekt wird eine `NewPost`-Message gesendet, anhand derer der `NewsMaster`-Actor reagieren, und einen Hinweis über die soeben gesendete Nachricht an alle Benutzer ausgeben kann. Zuletzt wird als Message an die eigene Instanz ein Objekt vom Typ `ResetSelectedNews` gesendet, um das `SessionVar`-Objekt zurückzusetzen.

Die `sendNewsMail`-Methode von MailComet

Die Methode `sendNewsMail` versendet die ausgewählte Nachricht an alle Abonnenten des Nachrichtenerstellers. Dabei stellt sie die üblichen E-Mail-Teile wie Empfänger, Absender, Betreff und Inhalt zusammen und übergibt diese an die Methode `CometMailer.sendMail`. Außerdem vermerkt sie in `emailsToSend`, an wieviel Empfänger die Nachricht zu versenden ist.

```
//Datei src/main/scala/com/lehrkraftnews/comet/MailComet.scala, Ausschnitt sendNewsMail:
```

```

private def sendNewsMail{
  val msg = selectedNewsVar.is
  val subscriptions = Subscription.findAll(By(Subscription.sourceId, msg.userId))
  val subscribers: List[User] = subscriptions.flatMap(_subscriberId.obj)
  val toAddresses: List[String] = subscribers.map(_email.is).removeDuplicates
  log.debug("E-Mail_to_" + toAddresses + ":\n" + msg.content.is + "\n")
  val bodyPlusTos: List[CometMailer.MailTypes] = PlainMailBodyType(msg.content.is) ::
    toAddresses.map(To(_))
  CometMailer.sendMail(
    Full(this), // Comet callback: Send confirmations hereto.
    From("news@lehrkraftnews.de"),
    Subject(
      "Lehrkraftnews_zu_" + msg.userId.obj.map(_fullCommaName).openOr("?") +
      "_gültig_bis_" + msg.getGermanDateString
    ),
    bodyPlusTos: _ *
  );
  emailsToSend += toAddresses.length
}

```

Listing 10.12: MailComet: sendNewsMail-Methode

Die lowPriority-Methode eines CometActors

Um auf Actor-Messages⁴¹ reagieren zu können, muss eine CometActor-Klasse die partielle Funktion `lowPriority` entsprechend implementieren. Diese wird in der Beispielanwendung wie folgt fürs Formularrücksetzen bzw. für Rückmeldungen über den Versandfortschritt definiert:

```

//Datei src/main/scala/com/lehrkraftnews/comet/MailComet.scala, Ausschnitt lowPriority:
override def lowPriority : PartialFunction[Any, Unit] = {

  case ResetSelectedNews =>
    selectedNewsVar(News.create.userId(User.currentUser.openOr(User.id)))
    reRender(false)
    partialUpdate(JsRaw("""${'#entrydate'}.datepicker({dateFormat:'dd.mm.yy'});_"""))

  case MailSendSuccess(address) =>
    partialUpdate(PrependHtml("addressesSuccess", Text(address) ++ (<br/>)))
    updateSendStatus

  case MailAddressFailure(address) =>
    partialUpdate(PrependHtml("addressesFail", Text(address) ++ (<br/>)))
    updateSendStatus
}

```

41 Der Versand von Messages wird im Anhang A.6 beschrieben.

```

case MailingFailure(exception) =>
  super.error(ExceptionReporting.getMessages(exception))
  partialUpdate(Show("messages")) //of default template
  showSendAborted
}

```

Listing 10.13: MailComet: lowPriority-Methode

Beim Versenden von Messages an Actor-Klassen wird bevorzugt auf case-Objekte (ohne Parameter) oder case-Klassen (mit Parametern) zurückgegriffen, da Fehler, zum Beispiel bei der Schreibweise, im Gegensatz zu Strings schon beim Kompilieren der Anwendung bemerkt werden können. Bei der Definition von case-Klassen und -objekten wird das Schlüsselwort `case` vorangestellt. Dadurch erhalten sie eine Factory-Methode, die es ermöglicht, Instanzen ohne das Schlüsselwort `new` zu erstellen. Wichtiger jedoch ist die Unterstützung für Pattern-Matching-Operationen mit dem `case` Operator, die sie dadurch erhalten.

Die `lowPriority`-Methode ist für vier Message-Typen definiert. Einer davon wird in der Datei `MailComet.scala` definiert:

```

//Datei src/main/scala/com/lehrkraftnews/comet/MailComet.scala, Ausschnitt ResetSelectedNews:
case object ResetSelectedNews

```

Listing 10.14: MailComet: case object für Reset-Message

Die Anweisung `this ! ResetSelectedNews` in der `doSave`-Methode (Listing 10.11) bewirkt also, dass der erste der in `lowPriority` definierten Fälle eintritt. Daraus resultierend wird `selectedNewsVar` wieder mit einem neuen „leeren“ `News`-Objekt belegt. Die `reRender`-Methode erwirkt ein erneutes Ausführen der `render`-Methode, die Formularfelder sind danach wieder leer. Abschließend wird die `partialUpdate`-Methode ausgeführt. Der im `JsRaw`-Objekt übergebene JavaScript-Code bewirkt eine erneute Zuweisung des Datumsauswahl-Skriptes an das Datumseingabefeld. Der Grund für die Realisierung dieses Schrittes mit Messages ist die Vermeidung von Redundanzen. Die gleiche Funktionalität muss an anderer Stelle (siehe Abschnitt 11.3.4) mithilfe von Messages realisiert werden, daher wird an dieser Stelle die Fähigkeit von Actor-Objekten genutzt, Messages an sich selbst zu senden.

Würde die Checkbox „Nachricht als E-Mail versenden?“ in Ansicht „Nachricht hinzufügen“ (Abbildung 10.1) gesetzt, wird die `sendMail`-Methode des `CometMailer`-Objekts mit allen für die Nachricht passenden gefundenen Abonnenten als Empfänger ausgeführt (Listing 10.12). Dabei handelt es sich um eine geringfügige Modifikation des in Lift integrierten `Mailer`-Objekts.^{42 43} `CometMailer` erlaubt es, der `sendMail`-Methode einen `CometActor`

⁴² Anhang A.8 bietet einen Überblick über die Modifikationen.

⁴³ `CometMailer` wird zusammen mit `Mailer` in der `Boot`-Klasse konfiguriert.

als Parameter `callback` zu übergeben. Ist der Mailversand an eine E-Mail-Adresse erfolgreich, sendet `CometMailer` eine `MailSendSuccess`-Message, welche die betreffende Adresse enthält, an den Callback-Actor. Wenn der Versand wegen einer Empfängeradresse scheitert, wird eine Message `MailAddressFailure` dorthin geschickt. Scheitert der Versand aus anderen Gründen, ist es eine Message `MailingFailure`. Diese drei Message-Typen sind als case-Klassen in `CometMailer` definiert. Der Callback-`CometActor` kann so aufgrund der empfangenen Messages Rückmeldungen über Erfolg und Misserfolg des E-Mail-Versandes anzeigen, während dieser stattfindet. Neben dem `CometActor` (`this`) erhält die `sendMail`-Methode als Parameter Objekte, die den Absender, den Betreff, die Zieladresse und den Inhalt der Nachricht bestimmen. Die Zählervariable `emailsToSend` wird abschließend auf die Anzahl der Empfängeradressen gesetzt.

Empfängt `MailComet` eine Message vom Typ `MailSendSuccess` oder `MailAddressFailure`, wird die enthaltene E-Mail-Adresse unter Verwendung der `partialUpdate`-Methode in das entsprechende `div`-Element im Template eingefügt. Anschließend wird die folgende Methode `updateSendStatus` aufgerufen. Wenn hingegen eine Message des Typs `MailingFailure` empfangen wird, wird die enthaltene Ausnahme mittels `super.error` gemeldet und der Abbruch des Versands in der ansichtspezifischen Statuszeile angezeigt.

Die `updateSendStatus`-Methode von `MailComet`

Die Methode `updateSendStatus` (Listing 10.15) verringert den Wert der Zählervariablen `emailsToSend` um eins, und nutzt sie, um eine Information über die Anzahl der noch zu versendenden E-Mails im dafür vorgesehenen `div`-Element zu platzieren. Um zu verdeutlichen, dass der Versandvorgang noch nicht abgeschlossen ist, wird das Ladeanzeigesymbol⁴⁴ eingeblendet. Sobald die letzte E-Mail versendet wurde, wird eine entsprechende Nachricht auf der Webseite angezeigt und das Ladeanzeigesymbol ausgeblendet.

```
//Datei src/main/scala/com/lehrkraftnews/comet/MailComet.scala, Ausschnitt updateSendStatus:
def updateSendStatus(){
  emailsToSend -= 1
  val (statusMessage, visiAction) = if (emailsToSend <= 0){
    ("E-Mail-Versand_Abgeschlossen", Hide(ajaxSpinner))
  } else {
    ("Noch_" + emailsToSend + "_E-Mails_zu_versenden...", Show(ajaxSpinner))
  }
  log.debug(statusMessage)
  partialUpdate(SetHtml("statusMessage", Text(statusMessage)))
  partialUpdate(visiAction)
}
```

⁴⁴ Es wird die gleiche Ladeanzeige wie für AJAX-Anfragen genutzt, (siehe Abschnitt 11.1). Der Aufruf an dieser Stelle findet statt, um nach dem Verlassen der Seite, bei anschließender Rückkehr, weiterhin über den evtl. nicht abgeschlossenen Vorgang zu informieren.

Listing 10.15: MailComet: updateSendStatus-Methode

10.2.2. Koordination von CometActors

Der CometActor im letzten Abschnitt funktioniert autonom, d.h. er liefert Informationen an den Benutzer zurück, die nur diesen betreffen. Verschiedene Benutzer können im System angemeldet sein und voneinander unabhängige MailComet-Instanzen beim Erstellen oder Bearbeiten von Nachrichten gebrauchen. Um Interaktion zwischen Benutzern herzustellen, muss es möglich sein, Messages zwischen CometActor-Instanzen zu verschicken.

In der Beispielanwendung soll nach der Erstellung einer neuen Nachricht ein Hinweis bei allen Besuchern der Lehrkraftnews-Seite erscheinen. Der für die Ausgabe dieser Nachrichten zuständige CometActor NewsReader wird wie folgt im default-Template eingebunden.

```
<!-- Datei src/main/webapp/templates-hidden/default.html, Ausschnitt latestNews: -->
<div id="latest_news" class="wide_width">
  <lift:comet type="NewsReader" name="news_reader"/>
</div>
```

Listing 10.16: Einbindung des CometActors NewsReader im default-Template

Die render-Methode von NewsReader soll bei dessen Initialisierung zunächst nichts ausgeben; die Variable latestMessagesXhtml wird deshalb mit einem leeren Text-Element initialisiert. Der Actor NewsReader besitzt eine private Methode updateTable. Sie wird ausgeführt, wenn er eine Message des Typs UpdateList mit einem News-Objekt empfängt. Weitere Fälle sind in seiner lowPriority-Methode nicht definiert. Innerhalb von updateTable wird aus dem Namen des Verfassers und dem Inhalt des empfangenen News-Objekts ein Link zusammengesetzt und dem NodeSeq-Objekt latestMessagesXhtml hinzugefügt. Der abschließende Aufruf der reRender-Methode fügt es in das default-Template ein.

```
package com.lehrkraftnews.comet

import com.lehrkraftnews.model._
import net.liftweb.http._
import scala.xml._
import com.lehrkraftnews.comet.NewsMaster._

/** Informiert Benutzer über neue Nachrichten, indem dieser NewsReader entsprechende Hinweise
 * in die Webseite über das default-Template einfügt.
 * @author Thomas Fiedler */
class NewsReader extends CometActor {
```

```

/**Enthält den XHTML-Code aller neuen Nachrichten seit Sessionbeginn. Wird bei jeder neuen
 * Nachricht erweitert.*/
private var latestMessagesXhtml: NodeSeq = Text("")

/**Gibt die in latestMessagesXhtml gesammelten Hinweise über neue Nachrichten aus.*/
def render = latestMessagesXhtml

/**Erstellt einen Link mit dem Namen der Lehrkraft und dem Inhalt der Nachricht, und gibt ihn
 * mit einer Länge von maximal 150 Zeichen an alle Benutzer (auch den Urheber) aus.*/
private def updateTable(news: News) = {
  val linkString = news.userId.obj.map(_fullCommaName).open_! +
    ":" + news.content.is

  latestMessagesXhtml +=
    Text("Neue_Nachricht:_") ++
    (<a href="{ /news/viewSingle/" + news.id.is}>
      {linkString.substring(0, Math.min(linkString.length, 150)) + "..."}
    </a>
    <br/>)

  reRender(false)
}

/**Wartet auf Messages und reagiert entsprechend.*/
override def lowPriority: PartialFunction[Any, Unit] = {
  case UpdateList(news: News) => updateTable(news)
}

/**Wird bei der Erstellung der Instanz ausgeführt, und meldet sie bei NewsMaster an.*/
override def localSetup = {
  NewsMaster ! SubscribeReader(this)
  super.localSetup()
}

/**Wird vor der Zerstörung der Instanz ausgeführt, und meldet sie bei NewsMaster ab.*/
override def localShutdown = {
  NewsMaster ! UnsubscribeReader(this)
  super.localShutdown()
}
}

```

Listing 10.17: com.lehrkraftnews.comet.NewsReader

Um die Instanzen von `NewsReader` zentral erreichbar zu machen, wird ein `LiftActor-Singleton` `NewsMaster` definiert, bei dem sich die einzelnen `CometActors` registrieren können. `CometActors` besitzen die Callback-Methoden `localSetup` und `localShutdown`. Diese wer-

den beim Initialisieren bzw. beim Herunterfahren eines CometActors ausgeführt.⁴⁵ Diese Methoden werden in NewsReader genutzt, um die Messages SubscribeReader bzw. UnsubscribeReader mit einer this-Referenz an NewsMaster versenden. Hierbei handelt es sich um eine Actor-spezifische Anwendung des Observer-Patterns [GoF94], wobei NewsMaster die Rolle des *Subjects* und NewsReader die des *Observers* einnimmt. Der Actor NewsMaster ist in folgendem Listing 10.18 dargestellt.

```

package com.lehrkraftnews.comet

import com.lehrkraftnews.model._
import net.liftweb.actor._

/**Verwaltet NewsReader-Instanzen, die sich über Messages
 * an- und abmelden können, und informiert sie über neue Nachrichten.
 * @author Thomas Fiedler */
object NewsMaster extends LiftActor {

  /**Message zur Anmeldung einer NewsReader-Instanz*/
  case class SubscribeReader(reader: NewsReader)

  /**Message zur Abmeldung einer NewsReader-Instanz*/
  case class UnsubscribeReader(reader: NewsReader)

  /**Message, informiert NewsReader-Instanzen über neue Nachricht*/
  case class UpdateList(news: News)

  /**Message, informiert NewsMaster über neue Nachricht*/
  case class NewPost(news: News)

  /**Liste der angemeldeten NewsReader-Instanzen*/
  private var observers: List[NewsReader] = Nil

  /**Wartet auf Messages über An- und Abmeldungen und neue Nachrichten und reagiert entsprechend.*/
  protected def messageHandler = {
    case SubscribeReader(rdr) => observers ::= rdr
    case UnsubscribeReader(rdr) => observers -= rdr
    case NewPost(news) => observers.foreach(_ ! UpdateList(news))
  }
}

```

Listing 10.18: com.lehrkraftnews.comet.NewsMaster

Dabei handelt es sich um ein LiftActor-Singleton-Objekt. Im Gegensatz zu einem Scala-Actor muss dieses nicht explizit gestartet werden. Nach seiner Initialisierung wartet dieses Objekt auf Messages, die ihm geschickt werden. Jeder Erhalt einer Message durch den

⁴⁵ Um einen CometActor herunterzufahren, kann ihm eine Message in der Form „CometActor ! ShutDown“ gesendet werden. Dadurch wird die exit-Methode von Actor ausgeführt.

`LiftActor` bewirkt einen Aufruf der partiellen Funktion `messageHandler`. Diese verwaltet eine `NewsReader`-Liste, der `Observer` hinzugefügt können, indem sie innerhalb einer `SubscribeReader`-Message an das `NewsMaster`-Objekt gesendet werden. Nach dem gleichen Prinzip entfernt eine `UnsubscribeReader`-Message den `Observer` aus der Liste.

Ein `MailComet`-Actor sendet in seiner `doSave`-Methode eine Message `NewPost` mit der neu erstellten Nachricht an `NewsMaster` (Listing 10.11). Beim Empfangen dieser Nachricht sendet `NewsMaster` an jedes sich in der Liste `observers` befindliche `NewsReader`-Objekt die Message `UpdateList` mit der neuen Nachricht, woraufhin jeweils dessen bereits erläuterte Methode `updateTable` einen Hinweis auf die neue Nachricht ausgibt.

11. Die Boot-Klasse

Die `Boot`-Klasse aus dem Paket `bootstrap.liftweb` beinhaltet die Methode `boot`, die beim Starten der Webanwendung einmalig ausgeführt wird. Sie dient der Konfiguration der Webanwendung sowie der Initialisierung von anwendungsrelevanten Singleton-Objekten. Für Entwickler ist es sehr angenehm, dass die Anwendung komplett in compiliertem Scala-Code statt in unleserlichem XML konfiguriert wird.

11.1. LiftRules

Für die Konfiguration steht das Objekt `LiftRules` zur Verfügung, welches über seine Methoden und Datenfelder angepasst werden kann. Im ersten Abschnitt der `boot`-Methode werden alle `LiftRules`-Einstellungen vorgenommen:

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt LiftRules:
LiftRules.addToPackages("com.lehrkraftnews")

LiftRules.early.append {
  _setCharacterEncoding("UTF-8")
}

//BEGIN(exceptionHandler)
val excHdlPF: LiftRules.ExceptionHandlerPF = {
  case (runMode, req, exc) => {
    val requestPurpose = ExceptionReporting.requestPurpose(req)
    log.debug("ExcHdlPF_executing_on_" + requestPurpose.name + "_request:" + req.path
      + "_with_exception:\n", exc)
    requestPurpose match {
      case ExceptionReporting.RequestPurpose.page =>
        S.error(ExceptionReporting.getMessages(exc)) //report immediately
      case _ => ExceptionReporting.set(exc) //stores for later reporting
    }
    ExceptionResponse(runMode, req, exc)
  }
}
LiftRules.exceptionHandler.prepend(excHdlPF)
//END(exceptionHandler)
```

```

LiftRules.ajaxStart = Full() => LiftRules.jsArtifacts.show("ajax_spinner").cmd)

LiftRules.ajaxEnd = Full() => LiftRules.jsArtifacts.hide("ajax_spinner").cmd)

LiftRules.setSiteMap(MenuInfo.dslSiteMap) //Avoid reevaluation of site map.

//BEGIN(rewrite)
LiftRules.statelessRewrite.append {
  case RewriteRequest(ParsePath(List("news", "viewSingle", newsId), _, _, _), _, _) =>
    RewriteResponse(List("news", "viewSingle"), Map("newsId" -> newsId))
}
//END(rewrite)

```

Listing 11.1: LiftRules-Konfiguration in bootstrap.liftweb.Boot.boot

Mit der Methode `addToPackages` wird der Name des Pakets festgelegt, in dem sich die Unterpakete `snippet`, `view` und `comet` befinden. Innerhalb dieser Unterpakete sucht Lift nach den Klassen entsprechender Ausprägung, wenn diese über XML-Elemente in Templates eingebunden werden.

Die Funktionen, die sich in der Liste `LiftRules.early` befinden, haben das `Request`-Objekt vom Typ `javax.servlet.http.HttpServletRequest` als Parameter und werden zu Beginn der Request-Verarbeitung auf dieses angewendet. In der Beispielanwendung wird auf diese Weise die Zeichenkodierung für die Interpretation der Request-Parameter festgelegt.

Partielle Funktionen in `LiftRules.exceptionHandler` werden von Lift aufgerufen, falls bei der Abarbeitung eines Requests eine Ausnahme aufgetreten ist. Ihnen wird dabei der *run mode* (Entwicklung oder Produktion usw.), der `HttpRequest` und die aufgetretene Ausnahme übergeben. Diese Musterteile können gleichzeitig als Selektionsmuster dienen. Näheres zur Ausnahmebehandlungsstrategie in Kapitel 12.

Bei `ajaxStart` und `ajaxEnd` handelt es sich um Datenfelder vom Typ `Box[() => JsCmd]`. Die `JsCmd`-Objekte erzeugen JavaScript-Code auf der Webseite, der beim Start bzw. am Ende eines Ajax-Requests ausgeführt wird. Die Zuweisungen im Listing 11.1 bewirken, dass das HTML-Element mit der Id `ajax_spinner`, bei dem es sich um eine animierte GIF-Grafik im `default`-Template handelt, über ein CSS-Attribut ein- bzw. ausgeblendet wird. Auf diese Weise kann der Benutzer eine visuelle Rückmeldung über die Dauer einer asynchronen Anfrage erhalten.

`LiftRules.setSiteMap` weist der Anwendung ein `SiteMap`-Objekt zu. In diesem kann die Menüführung der Anwendung und eine Kontrolle des Zugriffs auf die einzelnen Bereiche der Anwendung festgelegt werden. Die Erstellung eines solchen Objekts wird im Abschnitt „SiteMap“ in diesem Kapitel erläutert.

Es besteht die Möglichkeit, URLs umzuleiten und Bestandteile von ihnen als Parameter für Requests zu verwenden. Mit Rewrite-Regeln, die dem listenähnlichen Feld `LiftRules.statelessRewrite` hinzugefügt werden, können solche Umleitungen definiert werden. (siehe Abschnitt 11.4)

11.2. Mailer-Konfiguration

Die Benutzerverwaltung der Beispielanwendung baut auf den Trait `MetaMegaProtoUser` auf. Einige von ihm zur Verfügung gestellten Funktionen wie die Validierung von E-Mail-Adressen bei der Registrierung und das Zurücksetzen von Passwörtern nutzen für den E-Mail-Versand das Singleton-Objekt `Mailer` aus dem Paket `net.liftweb.util`. Für den Versand von in der Applikation erstellten Nachrichten wird eine Modifikation dieses Objekts, `CometMailer`, verwendet. Beide Objekte müssen so konfiguriert werden, dass sie sich mit einem SMTP-Server verbinden können. Zu diesem Zweck existiert in der Boot-Klasse die Methode `configMailer`, in der diese Konfiguration vorgenommen wird. Zuerst liest diese Methode mittels `MailProperties.getAll` die Datei `mail.properties` (Listing 5.3). Über die `System.setProperty`-Methode werden alle eingelesenen Properties als System-Properties gesetzt. Damit werden die Host-Adresse, die Art der Authentifizierung und die TLS-Unterstützung gemäß JavaMail-Konvention konfiguriert. Die Zuweisung eines `Authenticator`-Objekts für den Benutzernamen und das Passwort an das jeweilige `authenticator`-Attribut schließt die Konfiguration von `Mailer` und `CometMailer` ab.

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt configMailer:
```

```
private def configMailer() {  
  val mailPropertyEntries = MailProperties.getAll  
  mailPropertyEntries.foreach {  
    entry =>  
    System.setProperty(entry.getKey.toString, entry.getValue.toString)  
  }  
  val username = System.getProperty("mail.username")  
  val password = System.getProperty("mail.password")  
  val passwordAuthenticator = new Authenticator {  
    override def getPasswordAuthentication = {  
      new PasswordAuthentication(username, password)  
    }  
  }  
  val authenticator = Full(passwordAuthenticator)  
  Mailer.authenticator = authenticator  
  CometMailer.authenticator = authenticator  
}
```

Listing 11.2: Mailer-Konfiguration in `bootstrap.liftweb.Boot.boot`

Die Datei `mail.properties` wird durch folgendes Objekt `MailProperties` eingelesen und als Scala-Set geliefert:

```
//Datei src/main/scala/com/lehkrkraftnews/util/MailProperties.scala, Ausschnitt object:
object MailProperties {

  def getAll: scala.collection.Set[java.util.Map.Entry[Object, Object]] = _properties.entrySet

  def apply(key: String): String = _properties.getProperty(key)

  private val _properties = _load

  private def _load: Properties = {
    val homeDir = System.getProperty("user.home")
    val result = new Properties
    val fis = new FileInputStream(homeDir + "/mail.properties")
    try{
      result.load(fis);
    }finally{
      fis.close();
    }
    val necessaryProperties = List(
      "mail.smtp.starttls.enable", "mail.smtp.host", "mail.smtp.auth",
      "mail.username", "mail.password",
      "mail.teacher.1", "mail.teacher.2", "mail.student"
    )
    for(p<-necessaryProperties){
      _checkProperty(result, p)
    }
    result
  }

  private def _checkProperty(properties: Properties, key: String){
    val value = properties.getProperty(key)
    if(value==null){
      throw new MailPropertyMissingExc(key)
    }
  }

  class MailPropertyMissingExc(key: String)
  extends multex.Exc("Cannot_find_mail_property_{0}", key)
}
}
```

Listing 11.3: Objekt zum Einlesen der Mail-Properties

11.3. SiteMap

Das `SiteMap`-Objekt definiert die Struktur der Webanwendung. Es ermöglicht die Generierung von Menüs und die Definition von Zugriffsbeschränkungen. Weiterhin ist es möglich festzulegen, welches Template beim Aufruf einer URL verwendet werden soll. Trotz der sich bietenden Vorteile ist es nicht zwingend erforderlich, die Struktur der Webanwendung mithilfe des `SiteMap`-Objekts zu definieren. In diesem Fall kann auf jeden Ort der Webanwendung zugegriffen werden. Sobald jedoch ein `SiteMap`-Objekt definiert und registriert wird, *muss* jeder Pfad, der über eine URL erreichbar sein soll, explizit durch ein `Menu`-Objekt beschrieben werden.

Der Konstruktor von `SiteMap` erhält als optionale Parameter Objekte vom Typ `Menu`. Ein `Menu`-Objekt muss mit einem `Loc`-Objekt initialisiert werden. Der *Sequence Argument*-Markierer `:_*`⁴⁶ erlaubt es, alle Elemente einer Liste als einzelne, optionale Parameter zu übergeben. Die Methode `LiftRules.setSiteMap` registriert das `SiteMap`-Objekt für die Webanwendung.

```
def bereich2Menu =
  Menu(Loc("bereich2Sub1Loc", List("bereich2", "sub1"), "Sub1")) ::
  Menu(Loc("bereich2Sub2Loc", List("bereich2", "sub2"), "Sub2")) ::
  Nil

def exampleMenu =
  Menu(Loc("startLoc", List("index"), "Start")) ::
  Menu(Loc("bereich1Loc", List("bereich1", "index"), "Bereich1")) ::
  Menu(Loc("bereich2Loc", List("bereich2", "index"), "Bereich2"), bereich2Menu :_* ) ::
  Menu(Loc("bereich3Loc", List("bereich3") -> true, "Bereich3"), ) ::
  Nil

LiftRules.setSiteMap(SiteMap(exampleMenu :_*))
```

Listing 11.4: Beispiel: Erstellung einer einfachen `SiteMap`-Struktur

Listing 11.4 zeigt die Definition einer einfachen verschachtelten `SiteMap`-Struktur. Zunächst wird in der Methode `exampleMenu` eine Liste aus vier `Menu`-Objekten definiert. Jedem `Menu`-Konstruktor wird ein `Loc`-Objekt übergeben, das den Ort innerhalb der Webanwendung definiert.

Der erste Parameter weist dem Ort einen internen, frei wählbaren Namen zu. Der zweite Parameter ist eine Liste, die aus den, durch / getrennten, Elementen der URL besteht.

`List("bereich1", "index")` repräsentiert also den Pfad `/bereich1/index`.⁴⁷ Wird hier anstelle der Liste ein `Tuple2`-Objekt bestehend aus der Liste und dem Wert `true` überge-

⁴⁶ siehe §6.6 Scala Language Specification

⁴⁷ Die Dateinamensendung wird bei Templates automatisch hinzugefügt.

ben, wird der Pfad automatisch als Ordner verstanden, für dessen beinhaltete Elemente die gleichen Bedingungen gelten, wie für ihn. Im Beispiel kann auf alle Elemente innerhalb des Ordners „bereich3“ zugegriffen werden. Der dritte Parameter gibt an, unter welchem Namen der Menüpunkt auf der Webseite erscheinen soll.

Zusätzlich zum `Loc`-Objekt können einem `Menu`-Objekt in seinem Konstruktor weitere `Menu`-Objekte als optionale Parameter übergeben werden, welche als untergeordnete Orte interpretiert werden. Auf diese Weise können verschachtelte Menüs generiert werden. Außerdem ist es möglich, Zugriffsbedingungen, die für das Eltern-`Menu`-Objekt gelten, auf dessen Kind-Objekte zu übertragen. Dem mit „bereich2Loc“ benannten `Loc`-Objekt sind als Kind-Elemente die Elemente der in `bereich2Menu` definierten Liste zugewiesen.

11.3.1. Erweiterte Zugriffssteuerung

`SiteMap` bietet eine komfortable Möglichkeit, den Zugriff auf Orte an Bedingungen zu knüpfen. Zu diesem Zweck können `If`-Objekte definiert werden, die bei der Definition von `Loc`-Objekten als optionale Parameter hinzugefügt werden können. `If`-Objekte erhalten bei ihrer Erstellung als Parameter eine parameterlose Funktion mit dem Rückgabewert `Boolean`, die prüft, ob der Zugriff erlaubt sein soll. Der zweite Parameter `failMsg` des Typs `net.liftweb`.

`sitemap.FailMsg` dient als Fehlermeldung, die angezeigt wird, wenn der Zugriff verweigert wurde. Normalerweise erhält `failMsg` einen `String`. Es kann aber auch eine `XmlResponse` übergeben werden, wenn man z.B. Links einbetten will. Wird der Zugriff eines Eltern-`Menu`-Objektes an eine Bedingung geknüpft, gilt diese auch für seine Kind-Elemente. Im folgenden Beispiel ist der bedingungslose Zugriff nur auf „startLoc“ möglich. Alle anderen Orte können nur eingeloggte Benutzer besuchen. Einem `Loc`-Objekt können mehrere `If`-Objekte zugewiesen werden. Die durch sie definierten Zugriffsbedingungen werden dann in der angegebenen Reihenfolge nacheinander geprüft und müssen für einen Zugriff alle erfüllt sein.

```
val IfLoggedIn = If() => User.currentUser.isDefined, "Sie_sind_nicht_eingeloggt.")

def bereich2Menu =
  Menu(Loc("bereich2Sub1Loc", List("bereich2", "sub1"), "Sub1")) ::
  Menu(Loc("bereich2Sub2Loc", List("bereich2", "sub2"), "Sub2")) ::
  Nil

def exampleMenu =
  Menu(Loc("startLoc", List("index"), "Start")) ::
  Menu(Loc("bereich1Loc", List("bereich1", "index"), "Bereich1", IfLoggedIn)) ::
  Menu(Loc("bereich2Loc", List("bereich2", "index"), "Bereich2",
    IfLoggedIn, bereich2Menu :_*)) ::
  Menu(Loc("bereich3Loc", List("bereich3") -> true, "Bereich3", IfLoggedIn)) ::
  Nil
```

```
LiftRules.setSiteMap(SiteMap(exampleMenu :_*))
```

Listing 11.5: Beispiel: Zugriffssteuerung durch If-Objekte

Die in Listing 11.6 definierten Bedingungen aus der Beispieldanwendung prüfen folgende Fälle:

- `IfLoggedIn` prüft, ob der Benutzer eingeloggt ist.
- `IfIsTeacher` prüft, ob der Benutzer eingeloggt ist und die Rolle „Teacher“ besitzt.
- `IfIsStudent` prüft, ob der Benutzer eingeloggt ist und die Rolle „Student“ besitzt.
- `IfIsUser` prüft, ob der Benutzer eingeloggt ist und die Rolle „Student“ oder „Teacher“ besitzt (in Abgrenzung zu Administratoren).
- `IfIsAdmin` prüft, ob der Benutzer eingeloggt ist und die Rolle „Admin“ besitzt.

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt accessConditions:
/** Zugriffsbedingung: Benutzer ist eingeloggt. */
val IfLoggedIn = If() =>
  User.currentUser.isDefined, "Sie_sind_nicht_eingeloggt.")

/** Zugriffsbedingung: Eingeloggter Benutzer hat Rolle User.Teacher */
val IfIsTeacher = If() => User.loggedInAs(User.Teacher), "Sie_sind_keine_Lehrkraft!")

/** Zugriffsbedingung: Eingeloggter Benutzer hat Rolle User.Student */
val IfIsStudent = If() => User.loggedInAs(User.Student), "Sie_sind_kein_Student!")

/** Zugriffsbedingung: Eingeloggter Benutzer hat Rolle User.Teacher oder Rolle User.Student,
 * im Gegensatz zu Admin. */
val IfIsUser = If() => User.loggedInAs(User.Student, User.Teacher),
  "Sie_sind_weder_Student_noch_Lehrkraft!")
)

/** Zugriffsbedingung: Eingeloggter Benutzer hat Rolle User.Admin */
val IfIsAdmin = If() => User.loggedInAs(User.Admin), "Sie_sind_kein_Administrator!")
```

Listing 11.6: Definition der If-Objekte aus der Beispieldanwendung in `Boot.boot`

11.3.2. Die SiteMap-DSL ab Lift 2

Ab Lift 2.0 steht eine kompaktere Form der SiteMap-Definition parallel zu der vorstehend geschilderten, expliziten zur Verfügung. Sie macht insbesondere die Definition des durch ein Menu-Objekt zugreifbaren Pfades lesbarer. Diese Notation wird *SiteMap DSL*⁴⁸ genannt.

⁴⁸ DSL = Domain Specific Language

In bestimmten Fällen kommt man jedoch um die explizite Konstruktion eines Menu-Objekts mittels seines Konstruktors und des Loc-Konstruktors nicht herum. Beide Notationen können auch gemischt werden.

In der DSL wird ein Menüpunkt nach folgendem Muster definiert:

```
Menu("Link-Text") / "pathPart1" / "pathPart2" >> LocParam >> ...
```

Ein optionaler erster Parameter von Menu dient der applikationsweit eindeutigen Identifizierung eines Menüpunkts und entspricht damit dem ersten Parameter des Loc-Objekts. Die angehängten LocParam-Objekte können z.B. die Menügruppenzugehörigkeit oder Zugriffsbedingungen festlegen. Außerdem kann noch mittels submenu eine Liste von Untermenüpunkten angehängt werden.

In der Applikation zur Live-Demo⁴⁹ von Lift ist die SiteMap wie folgt definiert⁵⁰.

```
lazy val noGAE = Unless() => Props.inGAE, "Disabled_for_GAE")

def sitemap() = SiteMap(
  Menu("Home") / "index",
  Menu("Interactive_Stuff") / "interactive" submenu(
    Menu("Comet_Chat") / "chat" » noGAE,
    Menu("Ajax_Samples") / "ajax",
    Menu("Ajax_Form") / "ajax-form",
    Menu("Modal_Dialog") / "rhodeisland",
    Menu("JSON_Messaging") / "json",
    Menu("Stateless_JSON_Messaging") / "stateless_json",
    Menu("More_JSON") / "json_more",
    Menu("Ajax_and_Forms") / "form_ajax",
  Menu("Persistence") / "persistence" » noGAE submenu (
    Menu("XML_Fun") / "xml_fun" » noGAE,
    Menu("Database") / "database" » noGAE,
    Menu(Loc("simple", Link(List("simple"), true, "/simple/index"), "Simple_Forms", noGAE)),
    Menu("Templates") / "template" » noGAE),
  Menu("Templating") / "templating" / "index" submenu(
    Menu("Surround") / "templating" / "surround",
    Menu("Embed") / "templating" / "embed",
    Menu("Evaluation_Order") / "templating" / "eval_order",
    Menu("Select_<div>s") / "templating" / "selectomatic",
    Menu("Simple_Wizard") / "simple_wizard",
    Menu("Lazy>Loading") / "lazy",
    Menu("Parallel_Snippets") / "parallel",
    Menu("<head>_tag") / "templating" / "head"),
  Menu("Web_Services") / "ws" » noGAE,
  Menu("Localization") / "lang",
  Menu("Menus") / "menu" / "index" submenu(
```

49 <http://demo.liftweb.net/>

50 Quellcode unter <http://github.com/lift/lift/tree/master/examples/example>

```

Menu("First_Submenu") / "menu" / "one",
Menu("Second_Submenu_(has_more)") / "menu" / "two" submenus(
  Menu("First_(2)_Submenu") / "menu" / "two_one",
  Menu("Second_(2)_Submenu") / "menu" / "two_two"),
Menu("Third_Submenu") / "menu" / "three",
Menu("Forth_Submenu") / "menu" / "four"),
Menu(WikiStuff),
Menu("Misc_code") / "misc" submenus(
  Menu("Number_Guessing") / "guess",
  Menu("Wizard") / "wiz",
  Menu("Wizard_Challenge") / "wiz2",
  Menu("Simple_Screen") / "simple_screen",
  Menu("Arc_Challenge_#1") / "arc",
  Menu("File_Upload") / "file_upload",
  Menu(Loc("login", Link(List("login"), true, "/login/index"),
    <xml:group>Requiring Login<strike>SiteMap</strike> </xml:group>)),
  Menu("Counting") / "count"),
Menu(Loc("lift", ExtLink("http://liftweb.net"),
  <xml:group> <i>Lift</i></project home</xml:group>))
)

```

Listing 11.7: Beispiel: SiteMap-Definition in der Lift-Live-Demo-Applikation

Wir werden dies nicht genauer besprechen, sondern zeigen sofort die komplette SiteMap-Definition in unserer Beispiellapplikation, da sie einem etwas anderen Stil folgt. Hier werden keine Submenüs verwendet, sondern mittels LocGroup unterschiedene Gruppen von Menüpunkten. Zunächst also die Definition der Gruppen sowie eine Funktion zur Umwandlung einer Liste von Menüpunkten in ein SiteMap-Objekt.

```

//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt menuVorspann:
val mainGroup = LocGroup("main")
val newsGroup = LocGroup("news")
val userGroup = LocGroup("user")

SiteMap.enforceUniqueLinks = false
// :_* is a sequence argument marker. See Chapter 6.6 SLS:
val dslSitemap = SiteMap(dslMenuList: _*) //Avoid reevaluation of site map

```

Listing 11.8: Vorbereitung zur Menüdefinition in bootstrap.liftweb.Boot.boot

Sodann werden in der Methode dslMenuList in Listing 11.9 alle Menüpunkte der Beispiellapplikation definiert. Sie werden durchgängig in der DSL-Notation beschrieben, wobei man diese auch mit der Lift1-Notation mischen kann. Viele der Menüpunkte sind mittels >>mainGroup oder >>newsGroup einer der beiden Menügruppen aus Listing 11.8 zugeordnet. Der Zugriff auf viele Menüpunkte ist durch Konstrukte wie >>IfIsTeacher auf bestimmte Rollen eingeschränkt. Die Verwendung des Doppelsterns im Menüpunkt /Feed entspricht der -> true-Notation aus der traditionellen Sitemap-Definition.

Weitere Besonderheiten sind: An den durch `adminAccessLoc` identifizierten Menüpunkt werden alle durch `CRUDify[Long, News]` definierten Menüpunkte (`News.menu`) als Submenü angehängt. Am Ende werden noch alle in `createGroupedUserSitemap` definierten Menüpunkte an die Liste angehängt.

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt dslMenuList:
def dslMenuList = List(
  //BEGIN(mainMenu)
  Menu("startLoc", "Start") / "index" » mainGroup,
  Menu("newsMenuLoc", "Nachrichten") / "news" / "index" » mainGroup,
  Menu("subscriptionManageLoc", "Abonnements") / "subscriptions" / "manage"
    » mainGroup » IfLoggedIn » IfIsUser,
  Menu("helpLoc", "Hilfe") / "help" / "howto" » mainGroup,
  Menu("adminMenuLoc", "Admin") / "admin" / "users" / "index" » mainGroup » IfIsAdmin,
  Menu("throwExceptionLoc", "Throw") / "throw" » mainGroup,
  //END(mainMenu)

  //newsMenu:
  Menu("newsViewAllLoc", "Alle") / "news" / "index" » newsGroup,
  Menu("newsViewSubscribedLoc", "Abonnierte") / "news" / "viewSubscribed"
    » newsGroup » IfIsUser,
  Menu("newsViewByTeacherLoc", "Nach_Lehrkraft") / "news" / "viewByTeacher" » newsGroup,
  Menu("newsViewOwnLoc", "Eigene") / "news" / "viewOwn" » newsGroup » IfIsTeacher,
  //BEGIN(newsAddLoc)
  Menu("newsAddLoc", "Hinzufügen") / "news" / "add"
    » newsGroup » addNewsTemplate » IfIsTeacher,
  //END(newsAddLoc)

  //BEGIN(adminMenu)
  Menu("userAdminManageLoc", "Benutzer") / "admin" / "users" / "index" » IfIsAdmin,
  Menu("userAdminEditLoc", "Bearbeiten") / "admin" / "users" / "edit" » IfIsAdmin,
  Menu("adminAccessLoc", "HiddenAdminMenuAccessControl") / "admin"
    » IfIsAdmin submenu(News.menus: _*),
  //END(adminMenu)

  //nonMenuLocs:
  //BEGIN(newsViewSingleLoc)
  Menu("newsViewSingleLoc", "Detailansicht") / "news" / "viewSingle",
  //END(newsViewSingleLoc)
  //BEGIN(newsEditLoc)
  Menu("newsEditLoc", "Bearbeiten") / "news" / "edit" » IfIsTeacher,
  //END(newsEditLoc)
  Menu("newsDeleteLoc", "Löschen") / "news" / "delete" » IfIsTeacher,
  Menu("feedLoc", "Feed") / "Feed" / net.liftweb.sitemap.**
) ::: createGroupedUserSitemap
```

Listing 11.9: Menüdefinition in `bootstrap.liftweb.Boot.boot`

11.3.3. Generierung von Menüs

Für die Einbindung von durch SiteMap generierten Menüs in ein Template stehen mehrere Methoden des in Lift integrierten Menu-Snippets (`net.liftweb.builtin.snippet.Menu`) zur Verfügung.

Menu.builder

Um ein Menü aus der kompletten SiteMap-Struktur zu erzeugen, kann das Template-Tag `<lift:Menu.builder/>` verwendet werden. Dieses generiert als HTML eine ungeordnete Liste mit Links für alle Menu-Objekte, aus denen das SiteMap-Objekt zusammengesetzt wurde. Der aktuell besuchte Ort wird dabei nicht als Link sondern als ``-Element ausgegeben. Kind-Elemente werden mit verschachtelten Listen angezeigt, jedoch nur, wenn es sich bei der aktuell besuchten Seite um ein direktes Eltern- oder Geschwister-Element handelt. Desweiteren werden nur die Menüpunkte angezeigt, deren Zugriffsbedingungen erfüllt sind. Den einzelnen Elementen der Liste können Attribute zugeordnet werden.

```
<lift:Menu.builder ul:class="listclass"/>
```

Listing 11.10: Beispiel: Menü mit `Menu.builder`

Listing 11.11 zeigt beispielhaft die HTML-Struktur des durch `Menu.builder` generierten Menüs für einen eingeloggten Benutzer nach einem Klick auf „Sub2“.

```
<ul class="listclass">
  <li><a href="/index">Start</a></li>
  <li><a href="/bereich1/">Bereich1</a></li>
  <li><a href="/bereich2/">Bereich2</a>
    <ul class="listclass">
      <li><a href="/bereich2/sub1">Sub1</a></li>
      <li><span>Sub2</span></li>
    </ul>
  </li>
  <li><a href="/bereich3/">Bereich3</a></li>
</ul>
```

Listing 11.11: Beispiel: Generierte HTML-Liste für das Beispiel aus Listing 11.5

Wie bereits erwähnt, muss bei der Verwendung von SiteMap für jeden Ort der Webanwendung eine entsprechende Definition durch ein Menu-Objekt erfolgen. Dient ein Menu-Objekt lediglich der Zugriffserlaubnis für einen Pfad der Anwendung und soll nicht in dem durch

Menu.builder erzeugt Menüs erscheinen, kann bei der Definition seines Loc-Objekts das Hidden-Singleton-Objekt aus net.liftweb.sitemap.Loc übergeben werden:

```
Menu(Loc("beispielLoc", List("beispielpfad", "beispieltemplate"), "Beispiel", Hidden))
```

Listing 11.12: Beispiel: Ausschluss eines Ortes aus dem Menü durch Hidden

Menu.group

Umfangreiche Webanwendungen besitzen oft mehrere Menüs, die an unterschiedlichen Stellen im Layout angeordnet sind. Mithilfe von LocGroup-Objekten lassen sich Menu-Objekte zu Gruppen zusammenfassen, die dann gesondert durch die Methode Menu.group ausgegeben werden können. Abbildung 11.1 zeigt die Gruppenzugehörigkeit der drei Menüs aus der Beispielanwendung.

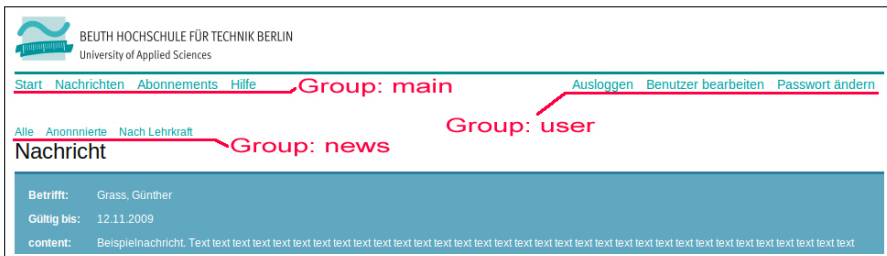


Abbildung 11.1.: Menü-Gruppen in der Beispielanwendung

Listing 11.13 zeigt die Definition des in der Menügruppe „main“ zusammengefassten Hauptmenüs der Beispielanwendung.

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt mainMenu:
Menu("startLoc", "Start") / "index" » mainGroup,
Menu("newsMenuLoc", "Nachrichten") / "news" / "index" » mainGroup,
Menu("subscriptionManageLoc", "Abonnements") / "subscriptions" / "manage"
  » mainGroup » IfLoggedIn » IfIsUser,
Menu("helpLoc", "Hilfe") / "help" / "howto" » mainGroup,
Menu("adminMenuLoc", "Admin") / "admin" / "users" / "index" » mainGroup » IfIsAdmin,
Menu("throwExceptionLoc", "Throw") / "throw" » mainGroup,
```

Listing 11.13: Definition der Menügruppe „main“ aus der Beispielanwendung

Zusammen mit dem Menü der Gruppe „user“ werden die Menüs mit dem vordefinierten Snippet `Menu.group` im Template `/templates-hidden/main_menu` eingebunden. Jeder Menü-Link wird nacheinander an die Stelle `<menu:bind/>` eingesetzt und mit einem ``-Element umrahmt. Im Gegensatz zu `Menu.builder` wird bei `Menu.group` nicht berücksichtigt, ob Menüpunkte mit dem Hidden-Objekt versehen sind. Es werden alle Elemente einer Gruppe angezeigt, deren Zugriffsbedingungen erfüllt sind.

```
<div>
  <ul style="float:left">
    <lift:Menu.group group="main">
      <li class="menuitem"><menu:bind/></li>
    </lift:Menu.group>
  </ul>
  <ul style="float:right;">
    <li class="menuitem"><lift:HomePageSnippet.eMailIfLoggedIn/></li>
    <lift:Menu.group group="user">
      <li class="menuitem"><menu:bind/></li>
    </lift:Menu.group>
  </ul>
  <div class="clear_div"/>
</div>
```

Listing 11.14: Erzeugung von Gruppenmenüs im Template `/templates-hidden/main_menu`

Der `MetaMegaProtoUser`-Trait stellt die Funktion `siteMap` zur Verfügung. Sie liefert eine Liste aller `Menu`-Objekte, die für die Benutzerverwaltung vom Trait `MetaMegaProtoUser` bereitgestellt werden (siehe Abschnitt 7.4). Um diese `Menu`-Objekte der Gruppe „user“ hinzuzufügen, wird mithilfe von Pattern-Matching⁵¹ überprüft, ob sie mit dem Hidden-Objekt versehen sind. Sind sie dies nicht, sind sie für die Anzeige in einem Menü vorgesehen. In diesem Fall wird das Objekt `userGroup` der Liste optionaler Parameter hinzugefügt.


```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt createGroupedUserSitemap:
def createGroupedUserSitemap: List[Menu] = {
  val userMenus = User.sitemap
  log.debug(userMenus.mkString("Generated_user_menus_are\n", "\n", ""))
  userMenus.map(_ match {
    case Menu(a) if !a.params.exists(_.equals(Hidden)) =>
      Menu(Loc(
        a.name, a.link.asInstanceOf[Link[Unit]], a.text.asInstanceOf[LinkText[Unit]],
        a.params.asInstanceOf[List[LocParam[Any]]] :: userGroup :: Nil
      ))
    case x => x;
  })
}
```

Listing 11.15: Hinzufügen von Menüpunkten aus `User.siteMap` zur Menügruppe „user“

⁵¹ Bei der `Menu`-Klasse handelt es sich um eine case-Klasse

Menu.item

Um einzelne Links zu Menüpunkten zu erstellen, kann die Methode `Menu.item` verwendet werden. Auf diese Weise werden in der Beispielanwendung die Menüpunkte „Benutzer“ und „Nachrichten“ in das Template `admin_menu` eingebunden (siehe Abbildung 11.2).



BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Start Nachrichten Hilfe Admin Ausloggen Benutzer bearbeiten Passwort ändern

Benutzer Nachrichten **admin_menu - Template**

Nachrichten

Nachricht erstellen

content	expirationdate	Lehrkraft			
Die Opel-Arbeitnehmer wollten eine Zukunft ... ehen bevor. Auch Politiker sind empört.	2009-11-28 00:00:00.0	1	View	Edit	Delete
Nach der Absage des Verkaufs von Opel an ... t der Autobauer bereits getilgt. mehr...	2009-11-21 00:00:00.0	1	View	Edit	Delete
Ein Brief, eine Podiumsdiskussion, ein T ... ge gut in ihren Reihen vorstellen. usw.	2009-11-26 00:00:00.0	1	View	Edit	Delete
Nachricht von Werner Wampe Blabiabla usw. usf.	2009-11-27 00:00:00.0	6	View	Edit	Delete

Abbildung 11.2.: Bereich "admin_menu" in der Beispielanwendung

```
<div>
  <lift:Menu.item name="userAdminManageLoc"/>
  <lift:Menu.item name="List_List(nachricht)"/>
</div>
```

Listing 11.16: Einbindung einzelner Menüpunkte in `/templates-hidden/admin_menu`

Listing 11.16 zeigt das Template. Mit dem `name`-Attribut wird der Name des `Loc`-Objektes angegeben, für welches ein Link erstellt werden soll. „List List(nachricht)“ ist der Name des von der `CRUDify`-Funktion `News.menus` erzeugten `Menu`-Objekts für die Auflistung aller vorhandenen Nachrichten. Um es zusammen mit den anderen von `CRUDify` bereitgestellten Menüpunkten nur für Administratoren zugänglich zu machen, wird im Ausschnitt `adminMenu` ein `Menu`-Objekt definiert, dessen Aufgabe lediglich darin besteht, als Eltern-Objekt für alle durch `News.menus` generierten `Menu`-Objekte zu fungieren und seine Zugriffsbeschränkungen dadurch an diese weiter zu geben.

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt adminMenu:
Menu("userAdminManageLoc", "Benutzer") / "admin" / "users" / "index" »IfsAdmin,
Menu("userAdminEditLoc", "Bearbeiten") / "admin" / "users" / "edit" »IfsAdmin,
Menu("adminAccessLoc", "HiddenAdminMenuAccessControl") / "admin"
  »IfsAdmin submenus(News.menus: _*),
```

Listing 11.17: Definition des Admin-Menüs in `Boot.boot`

Menu.title

Die Methode `Menu.title` kann verwendet werden, um den Linktext des aktuell besuchten Ortes auszugeben, wie er im dritten Parameter des `Loc`-Konstruktors bzw. als letzter Parameter von `Menu(...)` angegeben ist. Eine Anwendungsmöglichkeit ist die Erstellung eines dynamischen Titel-Tags im `<head>`-Bereich des `default`-Templates der Beispielanwendung:

```
<!-- Datei src/main/webapp/templates-hidden/default.html, Ausschnitt title: -->
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:lift="http://liftweb.net">
  <head>
    <title>lehrkraftnews : <lift:Menu.title/></title>
```

Listing 11.18: Erstellung dynamischer Seiten-Titel mit `Menu.title` im `default`-Template

11.3.4. Festlegung des zu verwendenden Templates

Templates werden in Lift anhand der URL aufgesucht. Die URL `http://localhost:8080/news/edit` lädt das Template `webapp/news/edit.html`. Dieser Automatismus lässt sich mithilfe von *Template*-Objekten umgehen, die einem `Loc`-Objekt als optionale Parameter übergeben werden können.

Die Ansichten „Nachricht bearbeiten“ und „Neue Nachricht erstellen“ in der Beispielanwendung teilen sich das Template `news/edit.html`. Um eine Nachricht zu bearbeiten, muss zunächst ein „Bearbeiten“-Link aus der Ansicht „Eigene Nachrichten“ angeklickt werden. (Abbildung 11.3) Die Links für eine eigene Nachricht (Löschen, Bearbeiten) werden in der Methode

`NewsSnippet.showOwn` in einem geschachtelten, iterierten `bind` ähnlich zu `NewsSnippet.showAll` (Listing 9.9) auf folgende Art erstellt:

```
//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt ownAct:
"actions" -> {
  link("/news/delete", () => newsToDelete(news), Text("Löschen")) ++
  Text("_") ++
  link(
    "/news/edit",
    () => {
      selectedNewsVar(news)
      S.session.foreach(
        _.findComet("MailComet")
          .asInstanceOf[List[CometActor]]
          .foreach(_reRender(false))
      )
    }
  )
}
```

```

    },
    Text("Bearbeiten")
  )//link
}

```

Listing 11.19: Erstellung der Aktionen-Links in der Methode NewsSnippet.showOwn

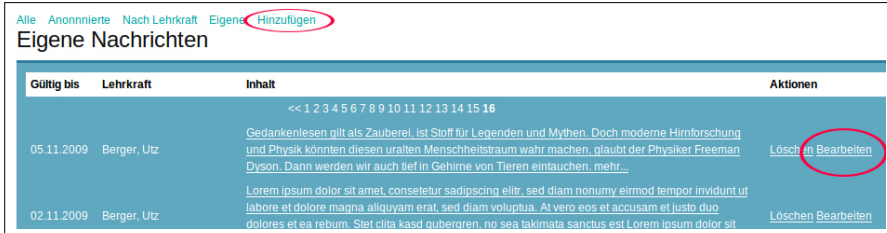


Abbildung 11.3.: Links „Bearbeiten“ und „Hinzufügen“ in der Ansicht „Eigene Nachricht“

Im Closure der zweiten link-Methode wird das SessionVar-Objekt `selectedNewsVar` mit dem zu bearbeitenden News-Objekt aus der Liste belegt. Durch

```

    ._findComet("MailComet")
    .asInstanceOf[List[CometActor]]
    .foreach(_reRender(false))

```

innerhalb von `S.session.foreach(...)` werden, falls eine Session vorhanden ist, alle Comet-Instanzen innerhalb der Session gesucht, die mit dem Namen „MailComet“ initialisiert wurden, und im Falle ihres Vorhandenseins zum Ausführen ihrer `reRender`-Methode bewegt. Auf diese Weise wird das Formular mit der zu bearbeitenden Nachricht gefüllt, die sich in `selectedNewsVar` befindet.

Der Ort `/news/edit` ist in einem Menu-Objekt im Abschnitt `newsEditLoc` definiert:

```

//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt newsEditLoc:
Menu("newsEditLoc", "Bearbeiten") / "news" / "edit" > IfIsTeacher,

```

Listing 11.20: Definition des Menu-Objekts für die Bearbeitung von Nachrichten

Da für die Erstellung von Nachrichten das gleiche Template benutzt werden soll, muss die zuständige CometActor-Instanz zunächst das `selectedNewsVar`-Objekt zurücksetzen und seine `reRender`-Methode ausführen. Anschließend soll die URI `/news/add` auf das Template

news/edit.html umgeleitet werden. Dazu kann ein Objekt vom Typ `Template` definiert werden.

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt addNewsTemplate:
val addNewsTemplate = Template(
  () => { S.session.foreach(_.findComet("MailComet").foreach(!_ ! ResetSelectedNews))
    (<lift:embed what="news/edit"/>)
  }
)
```

Listing 11.21: Erstellung eines Template-Objekts

Als Parameter in seinem Konstruktor erhält ein solches Objekt eine parameterlose Funktion die einen `NodeSeq`-Parameter zurückliefert. Mithilfe des `<lift:embed>`-Tags lässt sich auf diese Weise das Template `news/edit` einbetten, welches nun anstelle von `news/add` genutzt wird. Zuvor können im Körper der übergebenen Funktion beliebige Kommandos ausgeführt werden. Dies wird in der Beispielanwendung dazu genutzt, auf beschriebene Weise Comet-Instanzen mit dem Namen „MailComet“ in der Session zu suchen, und ihnen die Message `ResetSelectedNews` zu schicken, worauf `selectedNewsVar` zurückgesetzt, und die jeweilige Comet-Instanz neu gerendert wird (siehe Listing 10.13).

Der Ort `news/add` wird in einem Menu-Objekt definiert, das als zusätzlichen `LocParam` das in Listing 11.21 definierte `Template`-Objekt erhält:

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt newsAddLoc:
Menu("newsAddLoc", "Hinzufügen") / "news" / "add"
  »newsGroup»addNewsTemplate»IfIsTeacher,
```

Listing 11.22: Definition des Menu-Objekts für die Erstellung neuer Nachrichten

11.3.5. Übersicht der Menüstruktur der Beispielanwendung

Die Abbildung 11.4 bietet eine Übersicht über alle an der Definition des `SiteMap`-Objekts beteiligten Funktionen. Sie sind in das Objekt `MenuInfo` ausgelagert.

Für die durch `CRUDify` und `MetaMegaProtoUser` erstellten Menu-Objekte sind lediglich die Pfadangaben dargestellt, für alle anderen sind folgende Informationen abgebildet:

- der Name des `Loc`-Objekts
- der angezeigte Name im Menü
- der Pfad des durch sie definierten Ortes
- die durch Zugriffsbeschränkungen benötigte Rolle zum Besuch des Ortes

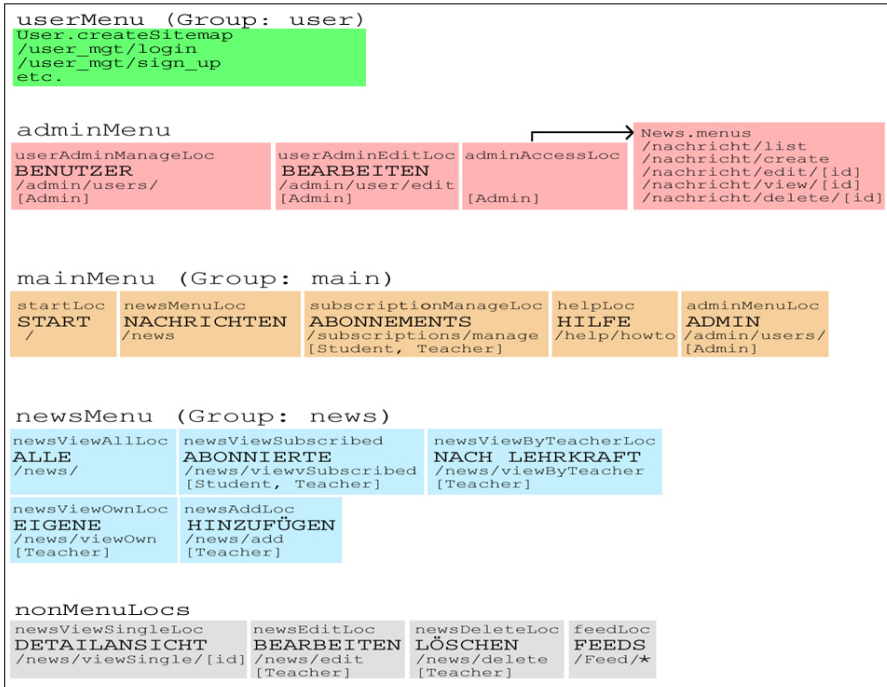


Abbildung 11.4.: Übersicht der Menu-Objekte der Beispielanwendung

11.4. URL-Rewriting

In der Beispielanwendung sollen einzelne Nachrichten direkt über eine eindeutige und leicht lesbare URL erreichbar sein. Die URL `/news/viewSingle/42` soll zu der Nachricht mit der ID 42 führen.

Im Menu-Objekt „`newsViewSingleLoc`“ wird der Ort für die Detailansicht definiert.

```
//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt newsViewSingleLoc:
Menu("newsViewSingleLoc", "Detailansicht") / "news" / "viewSingle",
```

Listing 11.23: Definition des Menu-Objekts für die Detailansicht von Nachrichten

Bei `LiftRules.statelessRewrite` handelt es sich um eine Sequenz aus partiellen Funktionen, die für Objekte vom Typ `net.liftweb.http.ReqwriteRequest` definiert sind und Objekte des Typs `RequestResponse` zurückgeben. Ein `RewriteRequest`-Objekt hat folgende Struktur:


```

case class RewriteRequest(
  val path: ParsePath,
  val requestType: RequestType,
  val httpRequest: HttpServletRequest
)

```

Folglich könnte man mithilfe des Platzhalterzeichens `_` die partielle Funktion für jedes erdenkliche `RewriteRequest`-Objekt definieren und auf den Pfad `/news/viewSingle` umleiten. Dem `RequestResponse`-Objekt kann dafür eine Liste der durch `/` getrennten URL-Elemente übergeben werden.

```

LiftRules.statelessRewrite.append {
  case RewriteRequest(_, _, _) =>
    RewriteResponse(List("news", "viewSingle"))
}
}

```

Listing 11.24: Umleitung aller Requests auf den Pfad `/news/viewSingle` (dadurch Endlosschleife)

Dies würde zu einer Endlosschleife führen, da die neue URL ihrerseits wieder umgeleitet werden würde. Das Beispiel dient lediglich zur Verdeutlichung des Pattern-Matching-Prinzips unter Benutzung des Platzhalters „`_`“. Eine wirksame Umleitung bedarf der Verfeinerung der Muster-Kriterien des `RewriteRequest`-Objekts. Da die Umleitung nur für URLs der Form `/news/viewSingle/*` stattfinden soll, muss der Parameter `path` vom Typ `ParsePath`, der den Request-Pfad definiert, angepasst werden. Objekte des Typs `ParsePath` besitzen folgende Struktur:

```

case class ParsePath(
  val partPath: List[String],
  val suffix: String,
  val absolute: Boolean,
  val endSlash: Boolean
)

```

Da die umzuleitende URL durch den Parameter `partPath` ausreichend eingeschränkt werden kann, kann für die restlichen Parameter von `ParsePath` wieder der Platzhalter `_` verwendet werden:

```

//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt rewrite:
LiftRules.statelessRewrite.append {
  case RewriteRequest(ParsePath(List("news", "viewSingle", newsId), _, _, _), _, _) =>
    RewriteResponse(List("news", "viewSingle"), Map("newsId" -> newsId))
}

```

Listing 11.25: Definition einer URL-Rewrite-Funktion in `Boot.boot`

Da die Angabe der Id hinter dem letzten / frei wählbar sein soll, wird sie in der den Pfad definierenden Liste als Variable `newsId` angegeben. Sie kann verwendet werden, um dem `RequestResponse`-Objekt ein `Map`-Objekt mit der Zuweisung von `newsId` an den String "newsId" zu übergeben. Dadurch wird die Variable innerhalb der Anwendung als Parameter-Variablen verfügbar gemacht. In der Wirkung entspricht dies einem Rewrite von z.B. `/news/viewSingle/5` nach `/news/viewSingle?newsId=5`. Der Zugriff erfolgt über die `param`-Methode des `S`-Objekts. Die weitere Funktionsweise der Ansicht für einzelne Nachrichten ist im Abschnitt 9.1 erklärt.

11.5. Konfiguration des Datenbankzugriffs

Im Abschnitt 7.3 (Seite 38) wird die Konfiguration des Zugriffs auf die Datenbank beschrieben, die in der `Boot`-Klasse vorgenommen wird.

11.6. Schemifier

Im Abschnitt 7.3 (Seite 38) wird erläutert, wie mit Hilfe des `Schemifier`-Objekts ein Datenbankschema anhand der definierten Modell-Klassen generiert werden kann. Der Aufruf der Methode `Schemifier.schemify` findet in der Methode `boot` der `Boot`-Klasse statt.

11.7. Initialisierung von Widget-Klassen

Durch die Ausführung ihrer `init`-Methode werden Widgets initialisiert. Dies sorgt unter anderem dafür, dass das `ResourceServer`-Objekt den Zugriff auf automatisch bereitgestellte Ressourcen freigibt. Im Fall von `TableSorter` handelt es sich um JavaScript-Bibliotheken für die Sortierung von Tabellen. `TableSorter.init` wird in der `boot`-Methode aufgerufen. Auf das `TableSorter`-Widget wird im Anhang A.5 näher eingegangen.

12. Ausnahmebehandlung und Zentrales Ausnahmemelden

In diesem Kapitel soll besprochen werden, wie man eine robuste und diagnosestarke Web-Applikation durch eine taugliche Strategie für die Ausnahmebehandlung erreicht.

12.1. Ausnahmebehandlung in der Anwendungssoftware

Das Ziel ist, dass es nirgendwo in der Applikation nötig sein soll, Ausnahmen explizit abzufangen. Gerade in Java-Code sieht man oft viel zu viel des Abfangens von Ausnahmen, da einen der Compiler quasi dazu zwingt, dies mit geprüften Ausnahmen zu tun.

Dies hat sich jedoch als problematisch erwiesen, weil die Anwendungsprogrammierer bei der Realisierung ihrer Funktionalität auch nicht genau wissen, was sie mit der abgefangenen Ausnahme machen sollten. Durch das Abfangen wird aber die normale Weitergabe der Ausnahme behindert und allzu häufig auch Diagnoseinformation unterdrückt.

Die durchgängig anzuwendende Strategie ist also, einmal erkannte Ausnahmen bei der Propagierung möglichst nicht zu behindern, eventuell mit weiteren Diagnoseinformationen anzureichern und dann auf oberster Ebene zentral dafür zu sorgen, dass sie in einem benutzerfreundlichen Format gemeldet werden. Für Entwickler sollten mehr Informationen gemeldet werden als für Normalbenutzer. Dieses Konzept nennen wir *Zentrales Ausnahmemelden*. Es ist genauer in einem Artikel des Java-Magazins beschrieben [KnHa07].

Der normale Programmierstil wäre es also, Ausnahmen nicht abzufangen, sondern in in das Zentrale Ausnahmemelden hinein laufen zu lassen.⁵²

Beim Zentralen Ausnahmemelden müssen wir in Lift verschiedene Szenarien unterscheiden, die in den nächsten Abschnitten behandelt werden: Full-Page-Requests, Ajax/Comet-Requests, in einem Comet-Actor, in einem Hintergrund-Actor.

⁵² In bestimmten Sonderfällen (Wiederholversuch, Dauerläufer) kann es dennoch nötig sein, Ausnahmen abzufangen und speziell zu behandeln.

12.2. Zentrales Ausnahmemelden bei Full-Page-Requests

Unter einem *Full-Page-Request* verstehen wir einen traditionellen HTTP-Request, der eine ganze HTML-Seite vom Server anfordert. Dies ist der Gegenbegriff zu einem Ajax/Comet-Request, der nur einen in die aktuell angezeigte Seite einzufügenden Teil liefert.

Um die Ausnahmebehandlung bei einem traditionellen Full-Page-Request testen zu können, ist im main-Menü der Applikation ein Menüpunkt „Throw“ eingebaut, der zur URL `/throw` führt. Diese ruft das Template `throw.html` auf, das im Folgenden abgedruckt ist:

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:lift='http://liftweb.net'>
  <head>
  </head>

  <body>
    <lift:NewsSnippet.throwException form="POST">
      Datum:
      <news:date id="entryform" />
    </lift:NewsSnippet.throwException>
  </body>
</html>
```

Listing 12.1: `src/main/webapp/throw.html`

Dieses wiederum ruft das folgende Snippet auf:

```
//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt throwException:
def throwException(xhtml: NodeSeq): NodeSeq = throw new Exception("Testweise provoziert")
```

Listing 12.2: `com.lehrkraftnews.snippet.NewsSnippet.throwException`

Standardverhalten von Lift

Wenn eine Ausnahme, die bei der Bearbeitung eines HTTP-Requests auftrat, bis hin zum Framework Lift propagiert, wird eine nicht besonders benutzerfreundliche Fehlerseite mit dem Stack Trace angezeigt, z.B. bei Anforderung der URL `/throw`:

```
Exception occured while processing/throw

Message: java.lang.Exception: Testweise provoziert
  com.lehrkraftnews.snippet.NewsSnippet.throwException(NewsSnippet.scala:318)
```

```

sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
java.lang.reflect.Method.invoke(Method.java:597)
net.liftweb.util.ClassHelpers$$anonfun$$net$liftweb$util$ClassHelpers
  $$__invokeMethod$$anonfun$apply$8.apply(ClassHelpers.scala:366)
...

```

Listing 12.3: Eine durch Lift angezeigte Standard-Fehlerseite

Der Ausnahmebehandlungsingriffspunkt in Lift

Man kann das Ausnahmebehandlungsverhalten bei Full-Page-Requests in Lift zentral konfigurieren. Die Sequenz `LiftRules.exceptionHandler` enthält partielle Funktionen mit der Signatur:

```
(net.liftweb.util.Props.RunModes.Value, net.liftweb.http Req, Throwable) => LiftResponse
```

Listing 12.4: Signatur der Ausnahmebehandler von Lift

Die wesentlichen `RunModes` dabei sind `Development`, `Test` und `Production`. Eine ganz primitive Fehlerseite würde bewirkt werden durch Folgendes in `Boot.boot`:

```

LiftRules.exceptionHandler.prepend {
  case (_, _, exc) => PlainTextResponse(exc.toString)
}

```

Listing 12.5: Einsetzen eines primitiven Exception Handlers in der Klasse `Boot`

Standardmäßig ist `LiftRules.exceptionHandler` wie folgt definiert:

```

@volatile var exceptionHandler = RulesSeq[ExceptionHandlerPF].append {
  case (Props.RunModes.Development, r, e) => XhtmlResponse(
    (<html> <body>Exception occurred while processing {r.uri}<pre>{showException(e)}</pre>
    </body> </html>),
    ResponseInfo.docType(r), List("Content-Type" -> "text/html;_charset=utf-8"),
    Nil, 500, S.ieMode
  )

  case (_, r, e) =>
    Log.error("Exception being returned to browser when processing_" + r, e)
    XhtmlResponse(
      (<html> <body>Something unexpected happened while serving the page at {r.uri}
      </body> </html>),
      ResponseInfo.docType(r), List("Content-Type" -> "text/html;_charset=utf-8"),

```

```

Nil, 500, S.ieMode
)
}

```

Listing 12.6: Der Standard-Ausnahmebehandler von Lift

Dies bedeutet, dass im Development-Modus eine Antwortseite wie in Listing 12.3 mit der angeforderten URI und dem Stack-Trace, aber ohne default-Template geliefert wird. In allen anderen Modi wird nur die URI angezeigt, aber dafür serverseitig eine Log-Meldung geschrieben.

Alle partiellen Funktionen in `LiftRules.exceptionHandler` werden beim Auftreten einer unbehandelten Ausnahme nacheinander durch Lift probiert. Da die in Listing 12.6 abgedruckte partielle Funktion mittels `case (_, r, e)` immer eingreift, müssen eigene Ausnahmebehandler vor diese eingefügt werden. Dies muss mit `LiftRules.exceptionHandler.prepend` geschehen.

Lösung in der Beispielapplikation

In der Beispielapplikation wird der zentrale Ausnahmebehandler wie folgt eingesetzt:

```

//Datei src/main/scala/bootstrap/liftweb/Boot.scala, Ausschnitt exceptionHandler:
val excHdlPF: LiftRules.ExceptionHandlerPF = {
  case (runMode, req, exc) => {
    val requestPurpose = ExceptionReporting.requestPurpose(req)
    log.debug("ExcHdlPF_executing_on_" + requestPurpose.name + "_request:" + req.path
      + "_with_exception:\n", exc)
    requestPurpose match {
      case ExceptionReporting.RequestPurpose.page =>
        S.error(ExceptionReporting.getMessages(exc)) //report immediately
      case _ => ExceptionReporting.set(exc) //stores for later reporting
    }
    ExceptionResponse(runMode, req, exc)
  }
}
LiftRules.exceptionHandler.prepend(excHdlPF)

```

Listing 12.7: Einsetzen eines zentralen Exception Handlers in der Klasse Boot

Dieser Ausnahmebehandler protokolliert zunächst die aufgetretene Ausnahme mittels `log.debug`. Sodann unterscheidet er die Fälle eines vollen Page- und eines Ajax/Comet/Actor-Requests. Bei einem Page-Request werden der Ausnahme entsprechende Meldungen mittels `S.error` für eine Ausgabe durch das default-Template vermerkt. Bei andersartigen Re-

quests kann die Ausnahme nicht sofort gemeldet werden und wird daher mittels `ExceptionReporting.set(exc)` für eine spätere Ausgabe vermerkt; siehe den folgenden Abschnitt 12.3. Am Ende wird eine HTML-Seite mit den Fehlerinformationen zusammengestellt und geliefert. Sie kommt allerdings nur bei Page-Requests zur Anzeige. Die Fehlerseite ist ein Objekt der case-Klasse `ExceptionResponse`. Deren `Factory`-Methode erhält den `runMode`, den `request` und die `exception` übergeben und stellt eine informative und benutzerfreundliche Fehlerseite zusammen. Die Klasse `ExceptionResponse` ist im folgenden Listing 12.8 abgedruckt.

```
//Datei src/main/scala/bootstrap/liftweb/ExceptionResponse.scala, Ausschnitt class:
case class ExceptionResponse(runMode: RunModes.Value, request: Req, exc: Throwable)
extends HeaderDefaults with NodeResponse {
  private val log = ExceptionResponse.log

  val docType = ResponseInfo.docType(request)
  val code = 500 //Internal Server Error
  override val renderInIEMode = S.ieMode
  val out: Node = {
    //TODO Must quote multex.Msg.getMessages(exc) as well as multex.Msg.getStackTrace(exc)
    //for HTML embedding. Knabe 10-03-16*/
    val excContent: Node = (<lift:surround with="default" at="content">
      <div><h1>Fehler aufgetreten</h1>
      <p>Wir konnten Ihre Anfrage <pre>{request.uri}</pre>
        wegen des oben gezeigten Fehlers nicht ausführen.
      </p>
      <h3>Der Fehler trat an folgender Stelle auf:</h3>
      <pre>{multex.Msg.getStackTrace(exc)}</pre>
      <table>
        <tr><th>RunMode:</th> <td>{runMode}</td></tr>
        <tr><th>Request:</th> <td>{request}</td></tr>
      </table>
    </div>
    </lift:surround>)
    log.info("excContent:_" + excContent)
    val requestBox = S.containerRequest
    log.info("requestBox:_" + requestBox)
    val result = requestBox match {
      case Full(httpRequest) =>
        //Applies Lift templating (for the surround) to excContent:
        S.render(excContent, httpRequest)(0) // (0) takes first Node of NodeSeq
      case _ =>
        excContent
    }
    log.trace("ExceptionResponse_result:_" + result)
    result
  }
}
```

Listing 12.8: Aufbau einer Fehlerseite in der Klasse `ExceptionResponse`

Die Schwierigkeit hierbei war es, eine Fehlerseite mit normalen Navigationsmöglichkeiten zusammenzustellen, d.h. die eigentliche Ausnahme mit dem default-Template zu umgeben. Lift wendet auf den von einer `ExceptionHandlerPF` zurückgegebenen `XhtmlResponse` seinen Rendering-Mechanismus (`<lift:surround>` und `<lift:embed>`) nicht an. Wir fanden jedoch die undokumentierte Methode

```
S.render(xhtml: NodeSeq, httpRequest: HTTPRequest): NodeSeq,
```

um dieses zu bewerkstelligen.⁵³ Leider funktioniert diese nur bei einer ROOT-Web-Applikation und nicht bei einer unter einem tieferem Pfad wie z.B. `/lehrkraftnews` anzuschreibenden.

Eine auf diese Weise zusammengestellte Fehlerseite nach absichtlichem Auslösen einer Ausnahme sieht dann wie folgt in Abbildung 12.1 aus. Man sieht oben die durch das default-Template bereitgestellte Navigation und rote Fehlermeldung sowie unten den Stack Trace der Ausnahme.

Einschränkungen

`LiftRules.exceptionHandler` wird durch Lift nur bei der Bearbeitung traditioneller `http-Requests`, die eine volle XHTML-Seite anfordern, sinnvoll berücksichtigt. Die von dort abgelegten Fehlerbehandlungsfunktionen zurückgegebenen XHTML-Inhalte werden nicht angezeigt, wenn AJAX- oder Comet-Requests bearbeitet werden, da diese Request-Typen XHTML nicht verarbeiten können. Sie erwarten z.B. eine Antwort, die ein JavaScript-Kommando enthält. Wie das Ausnahmemelden bei derartigen Requests zentralisiert werden kann, wird in den folgenden Abschnitten beschrieben.

12.3. Zentrales Ausnahmemelden bei Ajax-Requests

Ausnahmen können auch auftreten, wenn Lift einen AJAX-Request bearbeitet. Diese lassen sich durch `LiftRules.exceptionHandler` zwar abfangen, aber nicht melden.

Standardverhalten von Lift

Wenn eine Ausnahme, die bei der Bearbeitung eines AJAX-Requests auftrat, bis hin zum Framework Lift propagiert, wird diese protokolliert, wenn das Logging richtig konfiguriert ist. Der Bediener erhält im Browser keine Fehlermeldung.

⁵³ <http://www.mail-archive.com/liftweb@googlegroups.com/msg06047.html>

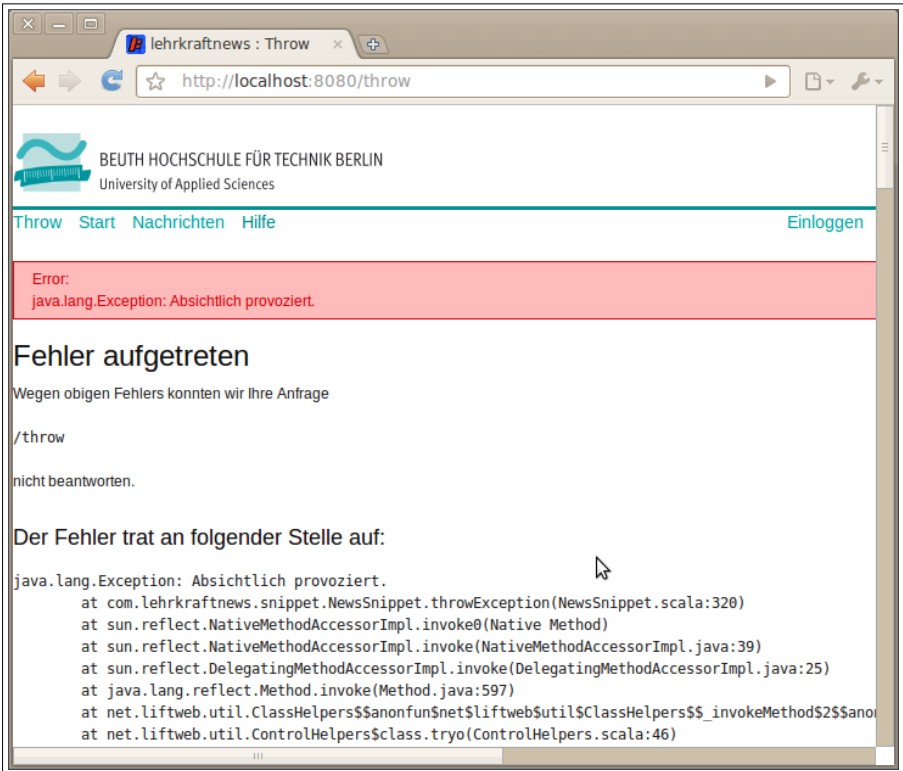


Abbildung 12.1.: Fehlerseite nach dem Provozieren einer Ausnahme mittels Menüpunkt *Throw*

Wenn man länger wartet, erscheint ein Bestätigungsdialog mit dem Text

„The server cannot be contacted at this time.“.

Aus den Protokolldateien kann man ersehen, dass der Klient den Request während der Wartezeit ungefähr dreimal wieder versucht. Die Fehlermeldung ist direkt irreführend, da sie den Bediener denken lässt, es gäbe Probleme mit der Netzwerkverbindung.

Wenn der Bediener aus Ungeduld während der Wartezeit auf einen Link klickt, geht ihm jeglicher Hinweis auf einen Fehler verloren.

Ein nutzbarer Eingriffspunkt in Lift

`S.addAround(lw: LoanWrapper)` bietet einen Eingriffspunkt für eine Vor- und Nachbearbeitung der Request-Verarbeitung durch Lift. Dabei werden alle Request-Arten abgefangen,

nämlich sowohl traditionelle Full-Page-Requests als auch die neuartigen AJAX- und Comet-Requests.

Dies ist neben `LiftRules.exceptionHandler` ein weiterer Ort, wo man Ausnahmen, die bei der Verarbeitung von AJAX-Requests auftreten, abfangen kann.

Wenn man diese dem Bediener melden will, so gestaltet sich dieses deutlich schwieriger. Der Klient erwartet eine Antwort in einem spezifischen Format, welches keine Ausnahme oder Fehlermeldung enthält.

Lösung in der Beispielapplikation

Wenn wir bei einem AJAX-Request eine Ausnahme abfangen, protokollieren wir sie und speichern sie für eine Anzeige bei der nächsten Anforderung einer vollen HTML-Seite. Wenn wir einen solchen Request erkennen, stellen wir eine der Ausnahme entsprechende Fehlermeldung mittels `S.error` in den Request, welche dann durch das `default`-Template angezeigt wird. Danach löschen wir die gespeicherte Ausnahme.

Auf diese Weise können wir sicher sein, dass der Bediener spätestens, wenn er mangels Antwort ungeduldig zu klicken anfängt, über den Fehler informiert wird.

Im Folgenden wird die Umsetzung in der Beispielanwendung beschrieben:

In der Methode `Boot.boot` rufen wir `S.addAround(ExceptionReporting)` auf. Dieses setzt das Objekt `ExceptionReporting` als *loan wrapper* ein. Die Verarbeitung eines Requests jeder Art durch `Lift` wird dadurch eingefasst in das, was in dem Wrapper beschrieben ist.

Das Objekt `ExceptionReporting` hat eine Methode `apply`, die eine Closure `doLiftProcessing` übergeben bekommt. In `apply` programmiert man die eigene Behandlung um einen Aufruf von `doLiftProcessing` herum. `apply` ist im folgenden Listing 12.9 abgedruckt.

```
//Datei src/main/scala/bootstrap/liftweb/ExceptionReporting.scala, Ausschnitt apply:
override def apply[T](doLiftProcessing: => T): T = {
  val requestPurpose = this.requestPurpose(S.request)
  log.debug("Before_" + requestPurpose.name + "_request_" + S.uri + "_by_" + S.request)
  if(requestPurpose == RequestPurpose.page){ //Traditional full HTML page request
    //Now report the last unreported exception, if exists:
    var oldExceptionMsg = takeOut()
    oldExceptionMsg match {
      case Empty =>
      case Full(messages) => {
```


wählen Sie die Menüpunkte *Nachrichten > Nach Lehrkraft*. In der Auswahlbox *Betreffend die Lehrkraft*: wählen die die Lehrkraft [*Inexistent*], die eigens für das Provozieren von Fehlern eingetragen ist. Ihr entspricht kein Datensatz in der Datenbank. Sodann klicken Sie auf den Button *Abschicken*.

Wenn alles gut ginge, würde mittels eines AJAX-Requests nur die im unteren Fensterbereich angezeigte Nachrichtenliste für die ausgewählte Lehrkraft aktualisiert werden. Da aber die Verarbeitung des AJAX-Request durch eine Ausnahme abgebrochen wurde, erhält der Bediener keine Antwort auf seine Anfrage. Erst nach einer längeren Zeit erscheint die Meldung „The server cannot be contacted at this time.“. Egal ob sie davor oder danach auf irgendeinen Link klicken, der auch das default-Template anzeigt, es wird darin die zuletzt aufgetretene Ausnahme gemeldet. Die Situation nach *Nachrichten > Nach Lehrkraft > [Inexistent]* und nachfolgender Navigation zur Startseite ist in Abbildung 12.2 dargestellt.

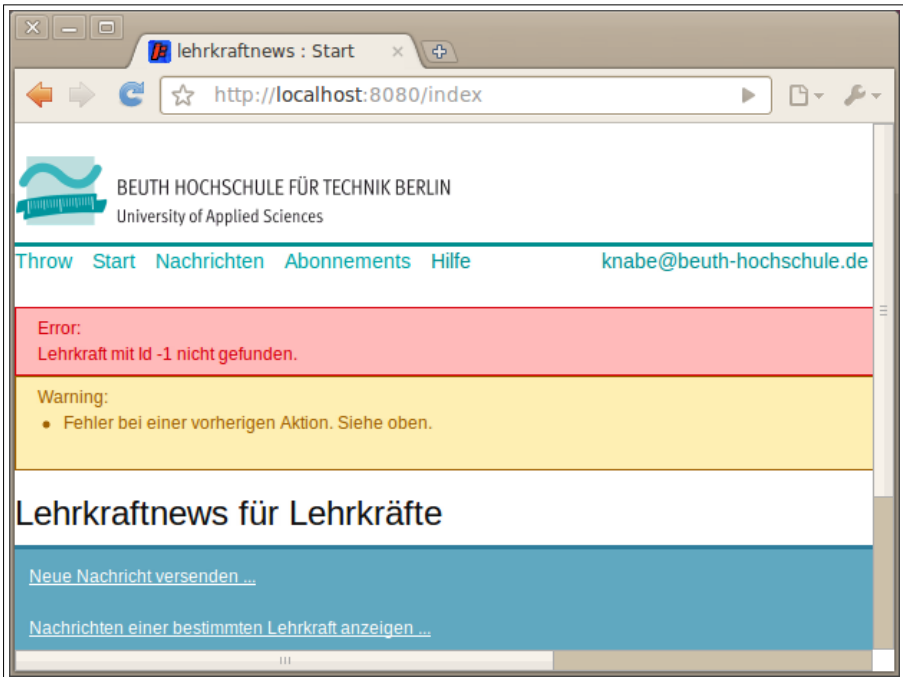


Abbildung 12.2.: Fehleranzeige nach dem Provozieren einer AJAX-Ausnahme und Folgenavigation

12.4. Zentrales Ausnahmemeldern bei Comet-Requests

Ausnahmen können auch auftreten, wenn Lift einen Comet-Request bearbeitet. Diese lassen sich durch `LiftRules.exceptionHandler` nicht melden.

Standardverhalten von Lift

Wenn eine Ausnahme, die bei der Bearbeitung eines Comet-Requests auftrat, bis hin zum Framework Lift propagiert, wird diese dem Benutzer nicht gemeldet, aber vom `net.liftweb.actor.ActorLogger` protokolliert. Wenn man länger wartet, erscheint auch hier ein Bestätigungsdialog mit dem Text

„The server cannot be contacted at this time.“.

Wenn der Bediener aus Ungeduld während der Wartezeit auf einen Link klickt, geht ihm der Hinweis auf den Fehler verloren.

Ein nutzbarer Eingriffspunkt in Lift

Die Klasse `CometActor` erbt eine Methode `exceptionHandler` von `SpecializedLiftActor`. Diese übernimmt das Protokollieren mittels des `ActorLoggers`. Durch Überschreiben dieser Methode kann man Ausnahmen, die bei der Verarbeitung von Comet-Requests auftreten, abfangen und sofort melden.

Lösung in der Beispielapplikation

Wenn wir in einem `CometActor` eine Ausnahme erkennen, geben wir mittels `warning` einen Hinweis darauf. Beachten Sie, dass dafür die `CometActor`-eigenen Meldungsmethoden statt derer aus dem `S`-Objekt verwendet werden müssen. Im Interesse eines robusten, zentralen Ausnahmemeldens lassen wir die Ausnahme jedoch in dieses hineinlaufen und erhalten die genaue Fehlermeldung bei der nächsten Anzeige einer vollen Seite durch die bei AJAX-Ausnahmen besprochene Methode `ExceptionReporting.apply` in Listing 12.9. Der Bediener wird darauf hingewiesen. Dieses Verfahren ähnelt stark dem bei AJAX-Ausnahmen.

Um dieses Verhalten zu zentralisieren, definieren wir einen `Trait DiagnosticCometActor` und überschreiben darin wie folgt die Methode `exceptionHandler`. Unsere Klasse `MailComet` beerbt diesen `Trait`.

```
//Datei src/main/scala/com/lehrkraftnews/util/DiagnosticCometActor.scala, Ausschnitt handler:
```

```

override def exceptionHandler = {
  case ex =>
    warning("""Wir konnten Ihren Comet-Request wegen eines Fehlers nicht ausführen.
    Die Meldung sehen Sie beim nächsten Klick.""")
    )
    partialUpdate(Show("messages")) //Aktualisiert <div id="messages"> im default-Template
  } //exceptionHandler

```

Listing 12.10: Methode exceptionHandler der Klasse DiagnosticCometActor

Provozierung einer Comet-Ausnahme

Um die Ausnahmebehandlung bei der Verarbeitung eines Comet-Requests zu demonstrieren, können Sie wie folgt vorgehen. Nach Durchführung der Testsuite melden Sie sich an mit dem entsprechend Listing 5.3 von Ihnen angegebenen Benutzernamen (E-Mail-Adresse) zu `mail.teacher.1`, d.h. für den Benutzer *Knabe, Christoph* mit Passwort `passwort`. Auf der erscheinenden Seite *Lehrkraftnews für Lehrkräfte* wählen Sie die Funktion *Neue Nachricht versenden* Geben Sie ein *Gültig bis*-Datum und unter *content* einen Nachrichtentext ein.

Wenn Sie jetzt den Button *Speichern* drücken würden, würde die Nachricht durch die Methode `doSave` der Klasse `MailComet` in der Datenbank gespeichert und an alle Abonnenten versendet werden. Um dabei eine Ausnahme zu provozieren, stellen Sie im Dateisystem bitte die Datei `target/database/H2.data.db` auf schreibgeschützt. Dann drücken Sie den *Speichern*-Button. Es wird durch den `DiagnosticCometActor` sofort ein Hinweis auf eine Ausnahme angezeigt, siehe Abbildung 12.3. Wenn man dann auf irgendeinen Link klickt, wird die genaue Ausnahme analog zu Abbildung 12.2 durch das `default-Template` angezeigt.

Auf Unix-Systemen kann es sein, dass sich das Schreibverbot erst nach dem nächsten Programmstart auswirkt. Sie können aber garantiert eine Ausnahme provozieren, wenn Sie als *content* die Zeichenkette `Unsinn` eingeben (siehe Prüfung in Methode `doSave` in Listing 10.11).

12.5. Ausnahmemelden bei Hintergrund-Actors

In vielen Applikationen gibt es langlaufende Hintergrund-Actors, die nicht direkt mit einer Oberflächenkomponente verbunden sind.

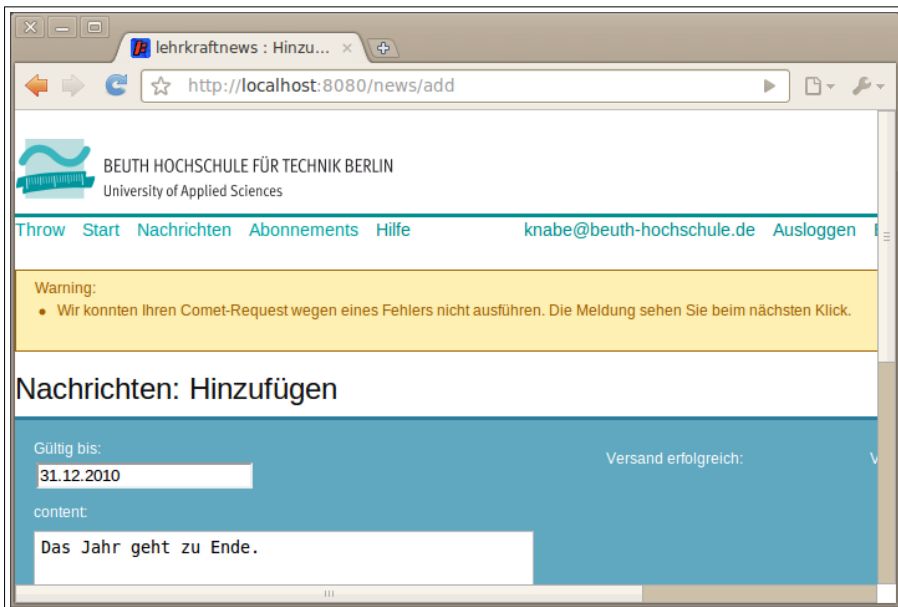


Abbildung 12.3.: Fehlerhinweis nach dem Provozieren einer Comet-Ausnahme

Typischerweise enthält deren `act`-Methode eine `loop` mit einem `react-case`-Block, in dem die verschiedenen Nachrichten, die der Actor akzeptiert, verarbeitet werden.

Standardverhalten eines Actors

Eine Ausnahme, die während der Ausführung des `react-case`-Blocks auftritt, unterbricht die `loop` und stoppt damit den Hintergrund-Actor. Der *Stack Trace* der Ausnahme wird auf `System.err` protokolliert.

Lösung in der Beispielapplikation

In dem `react-case`-Block fangen und melden wir alle Ausnahmen, um den langlaufenden Hintergrund-Actor nicht zu beenden.

In unserem Fall ist der langlaufende Actor eine eigene Modifikation von `net.liftweb.util.Mailer.MsgSender`. Da der Mailversand durch eine Bedieneraktion angestoßen wurde, melden wir jede Ausnahme in das anstoßende Formular, indem wir eine Fehlernachricht

zu dem CometActor senden, der mit dem Formular verbunden ist. Der CometActor muss dann die empfangene Fehlernachricht anzeigen. Er könnte dies durch einen Aufruf seiner eigenen Methode `error(message)` veranlassen. Dies würde eine Anzeige durch Lift als eine Fehlernotiz im `default`-Template bewirken.

Wir wählen hier jedoch eine interaktivere Form. Da es bei einer Vielzahl von Empfängeradressen oft Probleme mit nicht existierenden Adressen gibt, die als `javax.mail.SendFailedException` geworfen werden, sollen diese fehlerhaften neben den erfolgreichen Adressen auf der Eingabeseite gemeldet werden. Dazu wird eine `MailSendFailure`-Nachricht an den `CallbackCometActor` gesendet, welcher in unserem Fall der Actor `MailComet` ist. Dieser zeigt die erfolgreich und die fehlerhaft versandten Adressen in zwei Spalten fortlaufend an, wie es im Abschnitt 10.2 beschrieben und in Abbildung 10.1 gezeigt ist.

Wenn jedoch eine andere Ausnahme auftritt, hat der Mailversand allgemein versagt und sie soll wie zuvor besprochen als eine Fehlernotiz im `default`-Template angezeigt werden. Dazu sendet unser `MsgSender` eine `MailingFailure`-Nachricht an den Actor `MailComet`, welche diese ähnlich wie beim `DiagnosticCometActor` besprochen darstellt. Der für beide Ausnahμεarten zuständige `catch`-Zweig in `MailComet` ist nachfolgend abgedruckt.

```
//Datei src/main/scala/com/lehrkraftnews/util/CometMailer.scala, Ausschnitt catch:  
} catch {  
  case e: Exception => callback.foreach(c => {  
    c ! MailingFailure(e);  
    log.error("Failure_sending_mails:_", e)  
  });  
}
```

Listing 12.11: Ausnahmebehandlung im Hintergrund-Actor `CometMailer`

Eine dadurch erzeugte Fehlermeldung ist in Abbildung 12.4 zu sehen.

12.6. Zusammenfassung

Durch eine konsequente Zentralisierung allen Codes zur Meldung von Ausnahmen erhält man eine Applikation, die sich bei jedwedem Fehlern robust und definiert verhält. Im Rest der Applikation ist dann kaum mehr Ausnahmebehandlung nötig. Dabei lassen sich benutzerfreundliche Fehleranzeigen finden. Dies kann durch die Verknüpfung von Ausnahmen mit internationalisierbaren Meldungstexten noch wesentlich weiter als in der Beispielapplikation getrieben werden.

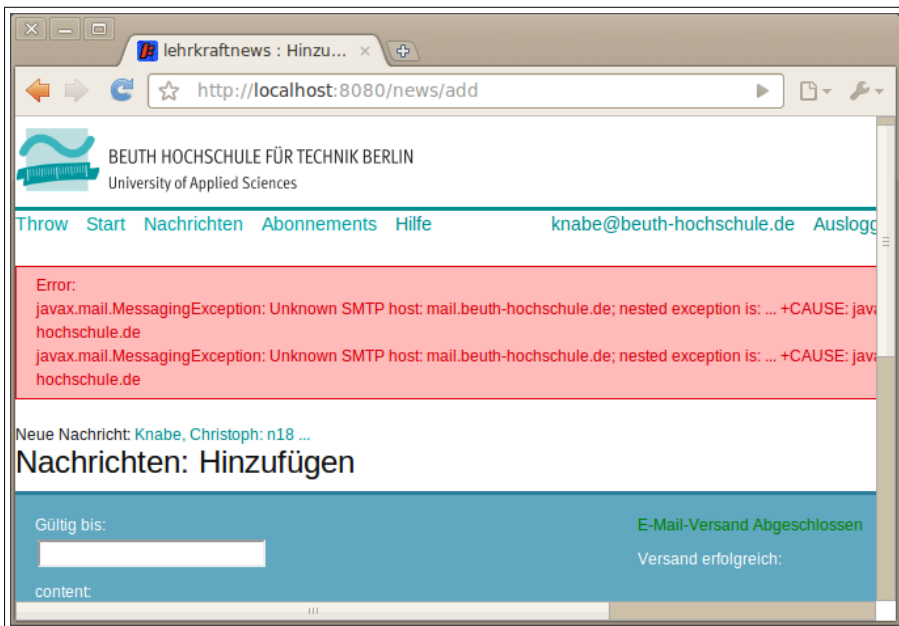


Abbildung 12.4.: Fehlermeldung nach einer Ausnahme in einem Hintergrund-Actor

13. Internationalisierung

Lift unterstützt die Erstellung mehrsprachiger Anwendungen. Dafür gibt es zwei Ansätze: die Verwendung internationalisierter Templates und die Erstellung von `.properties`-Dateien mit Übersetzungen von Zeichenketten.

13.1. Internationalisierte Templates

Die Möglichkeit, Mehrsprachigkeit durch entsprechend der Länderkennung benannte Templates zu erreichen, wurde bereits im Abschnitt 8.2.8 auf Seite 77 besprochen.

13.2. `properties`-Dateien

Eine weitere Möglichkeit stellt die Verwendung von `.properties`-Dateien dar. In ihnen können Nationaltexte definiert und jeweils einem Schlüssel zugeordnet werden.

13.2.1. `lift-core.properties`

Die in Lift integrierte Datei `lift-core.properties` enthält Nationaltexte, die an verschiedenen Stellen innerhalb des Frameworks ausgegeben werden. Die Methode `S.??` ersetzt den ihr übergebenen Schlüssel durch den entsprechenden Nationaltext.

```
def logoutMenuLoc: Box[Menu] =  
  Full(Menu(Loc("Logout", logoutPath, S.??("logout"),  
    Template(() => wrapIt(logout)),  
    testLogginIn)))
```

Listing 13.1: Definition des Menüpunktes „Ausloggen“ innerhalb des `MetaMegaProtoUser`-Traits.

Listing 13.1 zeigt die Definition des „Ausloggen“-Menüpunktes im Trait `MetaMegaProtoUser`. Die Verwendung von `S.??("logout")` anstelle eines festen Strings ermöglicht es, seine Benennung im Nachhinein innerhalb einer angepassten `.properties`-Datei zu ändern:

```
login = Einloggen
logout = Ausloggen
sign.up = Registrieren
Sign\u0020Up = Registrierung
lost.password = Passwort vergessen?
repeat = wiederholen
...
```

Listing 13.2: Ausschnitt aus der Datei `lift-core_de.properties` für deutsche Sprache

Die Core-Properties-Dateien befinden sich im Paket `i18n` im Ordner `src/main/resources`. Ihre Benennung erfolgt wie bei internationalisierten Templates durch das Anhängen der Länderkennung an das Datei-Präfix. Besitzt der Web-Browser eines Besuchers die Länderkennung „de“ wird innerhalb der Datei `lift-core_de.properties` nach einer Entsprechung für den Schlüssel „logout“ gesucht. Befindet sich keine solche Zuordnung innerhalb dieser Datei, wird stattdessen der Schlüssel ausgegeben. Dies führt dazu, dass Übersetzungen von `lift-core.properties` möglichst komplett erfolgen sollten.

Die originale `lift-core.properties` ist nicht immer konsequent strukturiert. Neben der durch Punkte getrennten Kleinschreibweise für die Schlüssel finden sich, durch Leerzeichen getrennte, Schlüssel in Großschreibweise. So existieren für die Zeichenkette „Registrierung“ sowohl die Kürzel „sign.up“ als auch „Sign Up“. Beide Formen kommen in Lift zum Einsatz und bedürfen daher einer Übersetzung. Um in einem Schlüssel innerhalb einer `.properties`-Datei ein Leerzeichen zu verwenden, muss dieses in Unicode-Schreibweise angegeben werden (`\u0020`).

13.2.2. lift.properties

Die Datei `lift-core.properties` ist für Nationaltexte vorgesehen, die im Rahmen von bereitgestellten Lift-Funktionalitäten verwendet werden. Für eigene Nationaltexte kann die Datei `lift.properties` verwendet werden. Eine solche Datei mit deutscher Länderkennung hat den Namen `lift_de.properties` und liegt direkt im Ordner `src/main/resources`. Nationaltexte aus dieser Datei können mit der Methode `S.?` eingebunden werden.

Verwendung in Templates

Um internationalisierte Zeichenketten innerhalb von Templates zu verwenden kann das Element `<lift:loc>` z.B. wie folgt verwendet werden.

```
<lift:loc locid="login">Einloggen</lift:loc>  
<lift:loc>login</lift:loc>
```

Listing 13.3: Zugriff auf internationalisierte Zeichenketten innerhalb von Templates

Das Attribut `locid` bezeichnet den Schlüssel aus der `.properties`-Datei. Wird kein entsprechender Nationaltext gefunden, wird der vom Element umschlossene Text ausgegeben.

Wenn kein `locid`-Attribut angegeben wird, wird der vom Element umschlossene Text implizit als Schlüssel verwendet und unverändert ausgegeben, wenn keine Entsprechung existiert.

14. Fazit

Das vorliegende Buch konnte einen Überblick über die wesentlichen Aspekte der Entwicklung von Webanwendungen mit Lift liefern. Es wurde demonstriert, wie der Aufwand für die Installation und Konfiguration von Lift-Projekten durch die Projektverwaltung mit Maven minimiert wird. Anhand der Beispielanwendung konnte gezeigt werden, welche Mittel Lift zur Verfügung stellt, um allgemeine Konzepte der Web-Entwicklung umzusetzen. Dabei wurde deutlich, dass die Realisierung des View-First-Patterns einen hohen Grad an Modularisierung und Wiederverwendbarkeit von Programmteilen ermöglicht, dadurch jedoch die strikte Trennung von Präsentations-, Steuerungs- und Geschäftslogik, wie sie in MVC-Frameworks vollzogen wird, aufgehoben wird. Die Datenhaltung mit dem Mapper-Framework ist komfortabel und typischer, bietet jedoch nur eine begrenzte Unterstützung für Relationen innerhalb der Datenbank und keine zusammengesetzten Attribute. Es besteht die Möglichkeit zur Formulierung eigener, komplexer SQL-Datenbankabfragen, der Preis dafür ist jedoch der Verlust der Unabhängigkeit vom verwendeten Datenbanksystem. Hervorzuheben sind die von Mapper bereitgestellten Funktionalitäten für die Benutzerverwaltung und die automatisierte Erstellung von Datenbankstrukturen und CRUD-Seiten. Sie ermöglichen die zügige Entwicklung von Prototypen und können dank umfassender Anpassungsmöglichkeiten darüber hinaus in fertigen Anwendungen Verwendung finden. Die Möglichkeit zur Erstellung zustandbehafteter Anwendungen erlaubt die komfortable Umsetzung komplexer Anwendungsfälle. AJAX- und Comet-Funktionalitäten werden von Lift hervorragend unterstützt und ihre Umsetzung kann mit wenig Aufwand betrieben werden.

Dem Umfang der Beispielanwendung wurde durch Lift an keiner Stelle Grenzen gesetzt. Es fanden sich für alle Problemstellungen Lösungen, deren Anpassung durch die enorme Flexibilität von Scala und Lift wunschgemäß erfolgen konnte. Lift erfüllt alle Voraussetzungen, um sich als Webframework zu etablieren. Eine Steigerung der Popularität von Scala durch Lift wie im Fall von *Ruby* und *Ruby on Rails* ist vorstellbar, da Lift einen konkreten Anwendungsfall für den Einstieg in Scala und die funktionale Programmierung bietet.

Ein großes Problem bei der Erstellung des vorliegenden Buches stellte die mangelhafte Dokumentation des Frameworks dar. Im noch einzigen erhältlichen Fachbuch⁵⁴ zum Thema Lift werden viele Themen nur oberflächlich behandelt. Die aufwändige Analyse des Lift-Quellcodes⁵⁵ war zur Informationsgewinnung unerlässlich, ebenso die Konsultierung der

⁵⁴ *The Definite Guide To Lift* von Derek Chen-Becker, Marius Danciu und Tyler Weir

⁵⁵ Dazu sollte man in der gewählten IDE Quelltext-Download aktiviert haben.

Lift-Mailing-Liste⁵⁶, zu deren Teilnehmern die Entwickler von Lift zählen. Zum Glück sind diese engagiert und hilfsbereit. Seit 2010 wird in dem Wiki bei Assembla⁵⁷ die gesamte Dokumentation zu Lift konsolidiert.

Die Entwicklung von Lift schreitet mit ungefähr monatlichen Meilensteinen aktiv voran. Mit 34 Committern hat Lift mittlerweile eine stabile Basis erreicht, die nicht mehr von einer Person abhängig ist. In 2012 wird mit dem Erscheinen von Lift in der Version 3.0 gerechnet. Aufgrund des andauernden Entwicklungsprozesses können an dieser Stelle keine konkreten Aussagen über Veränderungen in dieser Version gemacht werden. Es ist jedoch zu erwarten, dass das Persistenzframework *Record* dann neben *NoSQL*-Datenbanken auch eine Abbildung auf relationale Datenbanken ermöglicht.

⁵⁶ <http://groups.google.de/group/liftweb/>

⁵⁷ <http://www.assembla.com/wiki/show/liftweb>

A. Anhang

A.1. Bearbeitung der Beispielapplikation mit einer IDE

Es ist sehr zu empfehlen, den Quellcode der Beispielanwendung in eine der großen Java-Entwicklungsumgebungen wie *Eclipse* oder *IntelliJ IDEA* zu importieren. Da eine korrekte Beschreibung dafür schnell veraltet, haben wir sie in das Wiki des Assembla-Projektes Lift-Buch-Code ausgelagert.⁵⁸ Wenn es Ihnen gelingt, werden Sie mit Syntax Highlighting, Codenavigation, Refactoring und Debugging belohnt. Insbesondere bei schwach dokumentierten Bibliotheken ist ein Blick in den Quellcode dieser oft hilfreich. Dies ist aber nur bei funktionierender Codenavigation mit einem zumutbaren Aufwand verbunden.

Sie müssen allerdings damit rechnen, dass die Scala-Plugins noch nicht den Komfort und die Stabilität bieten, wie Sie das von den Java-IDEs gewohnt sind.

A.2. Logging in der Applikation mit SLF4J über Log4J

Seit Lift 2.0 protokolliert Lift seine internen Ereignisse über die leichtgewichtige Fassade *SLF4J* und bietet auch für die Lift-Applikationen entsprechende Dienste mittels des Traits `net.liftweb.common.Logger` an.⁵⁹ Standardmäßig meldet Lift auf die Konsole.

In der Beispielapplikation wird wegen der guten und bekannten Konfigurierbarkeit für die konkrete Protokollierung *Log4J* verwendet. Dieses muss seit Lift 2.2 explizit in der `pom.xml` angefordert werden (Listing A.1).

```
<!-- Datei pom.xml, Ausschnitt logging: -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.6.1</version>
</dependency>
```

⁵⁸ <http://www.assembla.com/wiki/show/liftbuchcode/IDE-Einrichtung>

⁵⁹ Beschrieben unter <http://www.assembla.com/wiki/show/liftweb/Logging>

```

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.6.1</version>
</dependency>
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.16</version>
</dependency>

```

Listing A.1: Anforderung von Logging mittels SLF4J über Log4J in pom.xml

Meldungen der Applikation werden aufgrund der nachfolgend abgedruckten log4j-Konfigurationsdatei `src/main/resources/default.log4j.xml` in die Datei `target/lehrkraftnews.log` ausgegeben.

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="true">
  <appender name="appender" class="org.apache.log4j.RollingFileAppender">
    <param name="file" value="target/lehrkraftnews.log"/>
    <param name="MaxFileSize" value="1000KB"/>
    <!-- Keep one backup file -->
    <param name="MaxBackupIndex" value="10"/>

    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{ISO8601} [%t] %-5p %c %x - %m%n" />
    </layout>
  </appender>
</root>
  <priority value="DEBUG"/>
  <appender-ref ref="appender"/>
</root>
</log4j:configuration>

```

Listing A.2: Konfiguration des Loggings mittels log4j

Damit auch beim Durchlaufen von Testtreibern mit derselben Log4J-Konfiguration gearbeitet wird, wird diese im Objekt `com.lehrkraftnews.util.LogFactory` explizit angefordert mittels

```

//Datei src/main/scala/com/lehrkraftnews/util/LogFactory.scala, Ausschnitt setup:
Logger.setup = Full(Log4j withFile new URL("file:target/classes/default.log4j.xml"))

```

Listing A.3: Explizite Initialisierung von Log4J

Alle Log-Anweisungen in der Beispiellapplikation benutzen diese LogFactory.

A.3. Box, Full und Empty

Bei der Entwicklung mit Lift stößt man insbesondere im Zusammenhang mit Datenbank-abfragen häufig auf die Typen Box, Full und Empty. Bei der Box-Klasse handelt es sich um ein Konstrukt ähnlich der Option-Klasse in Scala, also um eine generische abstrakte Klasse, mit dem erbenden Singleton Empty und den Unterklassen Full und Failure. Eine Box hat, wenn ein Eintrag in der Datenbank gefunden wurde, den Typ Full und enthält den tatsächlichen Wert des gefundenen Eintrags. Wurde kein Eintrag gefunden, ist sie vom Typ Empty, also leer. Es ist auch möglich, eine Ausnahme, die für das Nichtfinden ursächlich war, in einer Failure-Box zu liefern. Dies scheint jedoch nicht systematisch eingesetzt zu werden. Das Box-Konzept ermöglicht es, flexibel auf das Vorhanden- und Nichtvorhandensein von Werten zu reagieren, ohne das Risiko von NullPointerExceptions zur Laufzeit einzugehen.

Mithilfe der Methode foreach kann eine anonyme Funktion auf den Inhalt einer Box angewendet werden. Ist die Box leer, wird die Funktion nicht ausgeführt. Ist sie voll, wird sie genau einmal angewendet. Wenn im folgenden Beispiel ein Benutzer mit der Id 2 existiert, wird sein Nachname ausgegeben, anderenfalls wird die Funktion nicht ausgeführt.

```
val user = User.find(2) //suche Benutzer mit der id 2
user.foreach(u => Log.info("Name:_" + u.lastName))
```

Listing A.4: Beispiel: Box und foreach

Mit der Methode map kann nach dem gleichen Prinzip ebenfalls eine anonyme Funktion auf den Inhalt der Box angewendet werden. Ist die Box vom Typ Full liefert sie als Ergebnis wiederum eine Box mit dem Ergebnis dieses Funktionsaufrufs, andernfalls Empty. Ist also im nächsten Beispiel die Box user vom Typ Empty, da kein Benutzer gefunden wurde, ist es auch die Box name. Befindet sich in der Box user ein User-Objekt (mit dem Nachnamen „Müller“), so wird die Box name mit String „Müller“ gefüllt. Die Ausgabe lautet in diesem Fall also entweder „Name: Full(Müller)“ oder „Name: Empty“.

```
val user = User.find(2) //suche Benutzer mit der id 2
val name = user.map(u => u.lastName.is)
Log.info("Name:_" + name)
```

Listing A.5: Beispiel: Box und map

Um an den Inhalt einer Box zu gelangen, gibt es die Methoden `open_!` und `openOr`. Ist man sich sicher, dass eine Box gefüllt ist kann mit der Methode `open_!` auf den enthaltenen Wert zugegriffen werden. Hierbei ist größte Vorsicht geboten, da im Falle einer leeren Box eine `NullPointerException` geworfen wird. Dieser Ansatz entspricht daher nicht dem oben beschriebenen Sinn des Box-Konzeptes, wonach das Auftreten von `null`-Werten vermieden werden soll.

```
val user = User.find(2) //suche Benutzer mit der id 2
val name = user.map(u => u.lastName.is)
Log.info("Name:_" + name.open_!)
```

Listing A.6: Beispiel: Box und `open_!`

Als bessere Alternative steht die `openOr`-Methode der `Box`-Klasse zur Verfügung, welcher als Argument ein Default-Wert des durch die `Box` verwalteten Typs übergeben wird. Ist die `Box` gefüllt, liefert `openOr` deren Inhalt, im anderen Fall den übergebenen Default-Wert zurück.

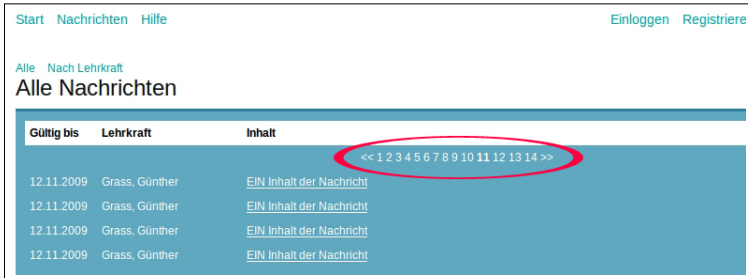
```
val user = User.find(1231) //suche Benutzer mit der id 1231
val name = user.map(u => u.lastName.is)
Log.info("Name:_" + name.openOr("Benutzer_nicht_gefunden"));
```

Listing A.7: Beispiel: Box und `openOr`

A.4. Paginierung

Um die Ausgabe von Nachrichten in den verschiedenen Ansichten übersichtlich zu gestalten, werden die Listen paginiert. Die Menge der angezeigten Nachrichten ist auf zehn begrenzt und es besteht die Möglichkeit, mithilfe von Links durch die dadurch entstehenden verschiedenen Seiten zu navigieren.

Zu diesem Zweck wurde innerhalb der `NewsSnippet`-Klasse das `Paginator`-Objekt (Listing A.8) definiert, das alle nötigen Methoden zur Erzeugung der Navigationsleiste enthält. Innerhalb der Anzeige-Snippets können die Werte `maxRows` und `offset` genutzt werden, um Steuer-Query-Parameter-Objekte (siehe Abschnitt 7.6.3) zu erstellen, die die Datenbankabfrage steuern (Listing A.9). Bei `maxRows` handelt es sich um einen konstanten Wert, die `offset`-Variable wird anhand eines an die URL angehängten GET-Parameters belegt. Der Zugriff auf den Parameter erfolgt durch die Methode `S.param`. Wird kein Parameter mit dem Namen „offset“ gefunden, erhält die Variable den Wert 0 und die ersten 10 Nachrichten werden ausgegeben.



Gültig bis	Lehrkraft	Inhalt
12.11.2009	Grass, Günther	EIN Inhalt der Nachricht
12.11.2009	Grass, Günther	EIN Inhalt der Nachricht
12.11.2009	Grass, Günther	EIN Inhalt der Nachricht
12.11.2009	Grass, Günther	EIN Inhalt der Nachricht

Abbildung A.1.: Paginierung bei der Nachrichtenaufistung

```
//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt Paginator:
object Paginator{

  /**Anzahl der maximal angezeigten Nachrichten pro Seite*/
  val maxRows = 10

  /**Gibt Position der ersten ausgegebenen Nachricht an, wird über die URL übergeben*/
  def offset = S.param("offset").map(_ toInt).openOr(0)

  /**Erstellt Links für Paginierung
   * @param numberOfEntries Anzahl der insgesamt anzuzeigenden Nachrichten
   * @return HTML-Links für "Vorherige" und "Nächste" sowie Seiten-Links */
  def paginate(numberOfEntries: Long): NodeSeq = {
    previousLink ++ pages(numberOfEntries) ++ nextLink(numberOfEntries)
  }

  /**Erstellt Seiten-Links
   * @param numberOfEntries Anzahl der insgesamt anzuzeigenden Nachrichten
   * @return HTML-Links für Seiten-Links */
  def pages(numberOfEntries: Long): NodeSeq = {
    val nop = (numberOfEntries / maxRows).toInt
    val numberOfPages = if (numberOfEntries % maxRows == 0) nop else nop + 1
    (1 to numberOfPages).map(p =>
      if (offset - p * maxRows + maxRows == 0) (<b>{p}&nbsp;</b>)
      else (<a href="{ "?offset="+((p-1)*maxRows).toString}>{p}&nbsp;</a>
    )
  }

  /**Erstellt Vorherige-Link
   * @param numberOfEntries Anzahl der insgesamt anzuzeigenden Nachrichten
   * @return HTML-Link für "Vorherige" */
  def previousLink: NodeSeq = {
    if (offset >= maxRows)
      (<a href="{ "?offset="+(offset-maxRows).toString}>{"<"&nbsp;</a>)
    else Text("")
  }
}
```

```

/**Erstellt Nächste-Link
* @param numberOfEntries Anzahl der insgesamt anzuzeigenden Nachrichten
* @return HTML-Link für "Nächste" */
def nextLink(numberOfEntries: Long): NodeSeq = {
  if (offset < numberOfEntries - maxRows)
    (<a href="{ "offset="+(offset+maxRows).toString}>{ "»"}</a>)
  else Text("")
}
}

```

Listing A.8: Das Singleton-Objekt Paginator

Die Erstellung der Links für die Navigation erfolgt anhand der Anzahl darzustellender Nachrichten. Diese kann in einer weiteren Datenbankabfrage mit der Funktion `count` ermittelt werden. Die Methode `Paginator.paginate` fasst die Erstellung der „Vor“- und „Zurück“-Links sowie die Generierung der Seiten-Links zusammen, und benötigt dafür als Parameter den zuvor ermittelten Wert. Die so erstellten Links werden mit der `bind`-Methode an der im Template dafür vorgesehenen Stelle `<news:pagination/>` platziert (Listing A.8).

```

//Datei src/main/scala/com/lehrkraftnews/snippet/NewsSnippet.scala, Ausschnitt showAll:
def showAll(xhtml: NodeSeq): NodeSeq = {
  var totalNumberOfNews = News.count
  val newsToShowOnOnePage: List[News] = News.findAll(
    OrderBy(News.expirationDate, Descending),
    MaxRows(Paginator.maxRows),
    StartAt(Paginator.offset)
  )
  bind("news", xhtml,
    "pagination" -> Paginator.paginate(totalNumberOfNews),
    "entries" -> newsToShowOnOnePage.flatMap( news => {
      bind("entry", chooseTemplate("news", "entries", xhtml),
        "content" ->
          (<a href="{ "/news/viewSingle/" + news.id.is.toString }>{ Text(news.content.is) }</a>),
        "date" -> Text(news.getGermanDateString),
        "person" -> Text(news.byUserId.obj.map(_fullCommaName) openOr "no_name")
      )
    }
  )
}

```

Listing A.9: Nutzung des Paginator-Objekts im `showAll`-Snippet

Bei der Erstellung der Links innerhalb der `Paginator`-Methoden `pages`, `previousLink` und `nextLink` wird der errechnete `offset`-Wert als GET-Parameter an die aktuelle URL angehängt.

Die Erstellung der Seitenlinks in der `pages`-Methode des Paginators kann dank der funktionalen Features von Scala sehr kompakt formuliert werden. Werte vom Typ `Int` besitzen die Methode `to` zur Erstellung von `Int`-Sequenzen. So liefert der Ausdruck `1.to(5)` die Sequenz `[1,2,3,4,5]`. Die Kurzschreibweise in Scala erlaubt die Formulierung `1 to 5`. Die `map`-Methode ermöglicht die Anwendung einer Funktion auf alle Mitglieder dieser Sequenz. Auf diese Weise wird in der Beispieldarstellung für jede Seite komfortabel ein Link erzeugt. Es wird zusätzlich geprüft, ob es sich bei der Seitenzahl um die aktuelle Seite handelt. Die Zahl wird in diesem Fall nicht als Link, sondern als fettgedruckter Text ausgegeben.

A.5. Das TableSorter-Widget

Lift beinhaltet JavaScript- und Scala-Bibliotheken für die vereinfachte Einbindung ansichtsbezogener Funktionalitäten in eine Anwendung. Zu diesen sogenannten Widgets zählen diverse Kalender-Ansichten sowie Widgets für die Darstellung von Baumstrukturen und Diagrammen. Ein weiteres nützliches Widget ist das `TableSorter`-Widget, welches für die Sortierung der Nachrichtenlisten in der Beispieldarstellung Verwendung findet. Die Sortierung erfolgt durch einen Klick auf den Kopf der zu sortierenden Tabellenspalte.

Um Lift-Widgets nutzen zu können müssen diese zunächst in der Maven-Projekt-Datei `pom.xml` importiert werden.

```
<!-- Datei pom.xml, Ausschnitt widgets: -->
<dependency>
  <groupId>net.liftweb</groupId>
  <artifactId>lift-widgets_${scala.version}</artifactId>
  <version>${lift.version}</version>
</dependency>
```

Listing A.10: Widgets-Abhängigkeit in `pom.xml`

Das `TableSorter`-Widget in Lift basiert auf dem jQuery⁶⁰-Plugin „TableSorter“ von Christian Bach (<http://www.tablesorter.com/>). Um sortiert zu werden, benötigt eine Tabelle im Document Object Model (DOM) der Webseite eine zugewiesene Id. Für die Ausgabe des notwendigen JavaScript-Codes wird in der Beispieldarstellung die `render`-Methode aus der Snippet-Klasse `TableSorterNewsSnippet` verwendet. Diese wird durch das Snippet-Element `<lift:TableSorterNewsSnippet tableId="news_table"/>` implizit aufgerufen. Die Id der zu sortierenden Tabelle wird ihr über das Attribut `tableId` mitgeteilt:

```
<!-- Datei src/main/webapp/news/index.html, Ausschnitt newsTable: -->
```

⁶⁰ jQuery ist eine freie JavaScript-Bibliothek die hauptsächlich Funktionalitäten für die DOM-Manipulation innerhalb von HTML-Seiten zur Verfügung stellt

```

<table id="news_table">
  <lift:TableSorterNewsSnippet tableId="news_table"/>
  <thead>
    <tr>
      <th>Gültig bis</th>
      <th>Lehrkraft</th>
      <th>Inhalt</th>
    </tr>
  </thead>
  <lift:NewsSnippet.showAll>
    <tr>
      <td class="pagination" colspan="3">
        <news:pagination />
      </td>
    </tr>
    <tbody>
      <news:entries>
        <tr>
          <td class="date_column"><entry:date /></td>
          <td class="person_column"><entry:person /></td>
          <td><entry:content /></td>
        </tr>
      </news:entries>
    </tbody>
  </lift:NewsSnippet.showAll>
</table>

```

Listing A.11: Verwendung des TableSorter-Widgets im Template news/index.html

Mit der `S.attr`-Methode kann auf den Wert des Attributs `tableId` aus dem Snippet-Element zugegriffen werden. Auf diese Weise ist es möglich, den JavaScript-Code so zu modifizieren, dass er auf die gewünschte Tabelle angewendet wird. Der Code verwendet das JavaScript-Framework jQuery, auf das in diesem Buch jedoch nicht näher eingegangen wird. Wesentliche Merkmale des Codes sind die Definition der Funktion `updateSorter` die durch den jQuery-Callback `ready` nach dem Laden der Seite ausgeführt wird. In ihm wird die Tabelle beim `TableSorter-jQuery`-Objekt angemeldet. Zudem wird festgelegt, dass die erste Spalte der Tabelle mit dem selbsterstellten Datums-Parser „`germanDate`“ sortiert wird, der in der Funktion `$.tablesorter.addParser` definiert wird, und die Sortierung von Kalenderdaten im deutschen Datumsformat erlaubt.

```

//Datei src/main/scala/com/lehrkraftnews/snippet/TableSorterNewsSnippet.scala, Ausschnitt class:
class TableSorterNewsSnippet{

  /**Id der zu sortierenden HTML-Tabelle.
   * Wird über Attribut "tableId" im Snippet-Element festgelegt*/
  val tableId = S.attr("tableId").map(_toString) openOr ("table")

```



```

/**JavaScript–Code der in die Webseite eingefügt wird.
 * Registriert die Tabelle über die Id beim tablesorter–jQuery–Objekt*/
val onLoad = { ""
  $(document).ready(function(){

  jQuery.fn.updateSorter=function(s){

  $(#' +s).tablesorter({
  headers:{0:{sorter:'germandate'}}
  });

  $.tablesorter.addParser({
  id:'germandate',
  is:function(s){
  return false;
  },
  format:function(s){
  var a=s.split('.');
  a[1]=a[1].replace(/^[0]+/g,"");
  return new Date(a.reverse().join("/")).getTime();
  },
  type:'numeric'
  });

  $(document).updateSorter('"+tableId+"');
  });
  ""
  }

/**Fügt die TableSorter–JavaScript–Bibliothek und den Code zur Registrierung der Tabelle
 * in den Head–Bereich der Webseite ein.
 *
 * @param xhtml Vom Snippet–Element im Template umgebene XML–Elemente
 * @return <script>–Elemente für die Einbindung von JavaScript
 */
def render(xhtml: NodeSeq): NodeSeq = {
  <head>
  <script type="text/javascript" src="/classpath/tablesorter/jquery.tablesorter.js"/>
  <script type="text/javascript">{onLoad}</script>
  </head>
  }
}

```

Listing A.12: com.lehrkraftnews.snippet.TableSorterNewsSnippet

Der modifizierte JavaScript-Code wird schließlich in der `render`-Methode zusammen mit der `Import`-Anweisung für die benötigte JavaScript-Bibliothek im `head`-Bereich der Webseite ausgegeben.⁶¹ Diese Bibliothek ist in Lift integriert. Um den Zugriff auf sie zu erlauben, muss das `TableSorter`-Widget jedoch einmalig initialisiert werden. Dies geschieht in der `boot`-Methode der Klasse `Boot` durch den Aufruf der Methode `TableSorter.init` (siehe Abschnitt 11.7).

Wird eine Tabelle durch Comet- oder Ajax-Datentransfer nach ihrer Registrierung nachträglich geändert, muss die JavaScript-Methode `updateSorter` erneut ausgeführt werden. In der Beispielanwendung ist dies in der Ansicht „nach Lehrkraft“ nach der Wahl der Lehrkraft nötig. In Listing 10.3 ist dieser Vorgang dargestellt.

A.6. Scala-Actors

Scalas `Actor`-Trait spielt bei der Umsetzung des Comet-Prinzips in Lift eine tragende Rolle. An dieser Stelle soll ein kurzer Überblick über *Actors* gegeben werden, um zum Verständnis für die Umsetzung des Comet-Prinzips in Lift beizutragen.

Mit `Actors` können nebenläufige Prozesse implementiert werden. Sie besitzen das Datenfeld `mailbox`, welches eine Art Postkasten für Nachrichten (*Messages*) darstellt. Durch das Versenden einer solchen `Message` kann mit `Actor`-Instanzen kommuniziert werden.

Um einen `Actor` zu implementieren, muss der Trait `scala.actors.Actor` beerbt werden, und die Methode `act` implementiert werden:

```
import scala.actors.Actor

object SimpleActor extends Actor {
  val words = "eins" :: "zwei" :: "drei" :: "vier" :: Nil
  def act() {
    words.foreach(word => {println(word); Thread.sleep(500)})
  }
}

object SimpleEnglishActor extends Actor {
  val words = "one" :: "two" :: "three" :: "four" :: Nil
  def act() {
    words.foreach(word => {println(word); Thread.sleep(500)})
  }
}

SimpleActor.start
```

⁶¹ siehe `Head Merging`, Abschnitt 8.2.9

```
SimpleEnglishActor.start
```

Listing A.13: Einfache Actors

Die Methode `start` startet einen Actor. Listing A.13 zeigt die Definition zweier einfacher Actors. Nach ihrem Start geben sie beide im Abstand von einer halben Sekunde die Wörter aus der Liste `words` aus. Da dies nebenläufig passiert, lautet die Ausgabe:

```
one
eins
two
zwei
three
drei
four
vier
```

Anschließend werden die Actor-Instanzen beendet.

Das folgende Beispiel zeigt einen Actor, mit dem durch die Versendung von Messages kommuniziert werden kann. Dazu kann die `react`-Methode verwendet werden, der eine partielle Funktion übergeben werden kann. Anhand dieser kann dann auf die Beschaffenheit der empfangenen Nachricht reagiert werden. Der Rückgabewert von `react` ist vom Typ `Nothing`. Das führt dazu, dass die Methode abrupt mit einer Exception endet. Für den Programmierer hat dies zur Folge, dass nur auf den Empfang von *einer* Nachricht reagiert werden kann. Es besteht jedoch die Möglichkeit, den Actor durch den erneuten Aufruf der Methode `act` wieder empfangsbereit zu machen. Messages können mit dem `!` Operator versendet werden.

```
import scala.actors.Actor

object GreetActor extends Actor {
  def act() {
    react { case "sayhello!" =>
      println("Hello!")
      act()
    }
  }
}

GreetActor.start
GreetActor ! "sayhello!"
GreetActor ! "sayhello!"
```

Listing A.14: Versand von Messages an einen Actor

`GreetActor` wartet nach seinem Start endlos auf Nachrichten. Lautet die Nachricht "say-hello!", gibt er „Hello“ aus. Lautet die Nachricht anders, erfolgt keine Reaktion; der Actor bleibt empfangsbereit.

Alternativ zum erneuten Aufruf von `act` nach dem Empfang der Nachricht kann der Aufruf von `react` der Funktion `loop` als Parameter übergeben werden, was zu seiner endlosen Wiederholung führt. Listing A.15 zeigt die Implementierung von `GreetActor` mit `loop`. Außerdem wird anstelle einer String-Message ein case-Objekt genutzt. Dies hat zum Vorteil, dass das statische Typ-System von Scala fehlerhafte Bezeichnungen von Messages schon zur Kompilierzeit melden kann.

```
import scala.actors.Actor
import scala.actors.Actor._

case object SayHello

object GreetActor extends Actor {
  def act(){
    loop{
      react { case SayHello =>
        println("Hello!")
      }
    }
  }
}

GreetActor.start
GreetActor ! SayHello
GreetActor ! SayHello
```

Listing A.15: case-Objekte als Message

Zuletzt soll die Möglichkeit demonstriert werden, mithilfe von case-Klassen Inhalte innerhalb von Messages zu versenden. Im Unterschied zu Objekten können Klassen Parameter besitzen, die für den Transport von Werten genutzt werden können:

```
import scala.actors.Actor
import scala.actors.Actor._

case class SayHelloTo(name: String)

object GreetActor extends Actor {
  def act(){
    loop{
      react { case SayHelloTo(name) =>
        println("Hello,_" + name)
      }
    }
  }
}
```

```
}  
}  
}  
  
GreetActor.start  
GreetActor ! SayHelloTo("Karl")  
GreetActor ! SayHelloTo("Inge")
```

Listing A.16: case-Klassen als Message

Die Ausgabe lautet:

```
Hello, Karl  
Hello, Inge
```

Lifts CometActors erben seit Lift 2 nicht vom Standard-Scala-Actor, sondern von einem LiftActor-Trait. Die Kommunikation über Messages erfolgt auf ähnliche Weise. Die Unterschiede sind in einem kurzen Newsgroup-Beitrag von David Pollak beschrieben.⁶²

A.7. Das S-Objekt

Zentraler Bestandteil von Lift ist das S-Objekt. Der Name leitet sich von State (Zustand) ab, da in diesem Objekt der Zustand der HTTP-Anfrage (Request) abgebildet wird. Durch seine Methoden bietet es die Möglichkeit, die HTTP-Antwort (Response) zu beeinflussen und Informationen über den Request zu erhalten, ohne ihn überall als Parameter durchzureichen. Eine Übersicht über die Felder und Methoden des Objekts soll an dieser Stelle nicht gegeben werden, stattdessen wird die Lektüre der Lift-API-Dokumentation zu diesem Objekt empfohlen. Anwendungen von S-Funktionalitäten in diesem Buch werden an den entsprechenden Stellen besprochen.

⁶² „We’ve migrated from Scala Actors to Lift Actors and included a series of traits that allow Lift to use its own Actors or Akka Actors (or anything else that implements that interface.) Lift no longer supports Scala Actors for Comet Actors. The GenericActor API offers pretty much the same client interface to Lift’s Actors, so ! and !? work the same way. However, there’s no link, self, start or exit methods.“ Siehe http://groups.google.com/group/liftweb/browse_thread/thread/824f14038bedf425

A.8. CometMailer

Das zum Versand von Nachrichten benutzte Objekt `CometMailer` basiert auf dem in Lift vorhandenen Objekt `Mailer`. Listing A.17 zeigt das originale `Mailer`-Objekt⁶³. In Listing A.18 bis A.23 sind die in `CometMailer` modifizierten Programmzeilen aufgeführt. Zusammen mit dem Programmcode im Repository kann sich der Leser so ein Bild von den Änderungen machen.

```
package net.liftweb.util

/* Copyright 2006–2008 WorldWide Conferencing, LLC
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License. */

import _root_.scala.xml.{NodeSeq}
import _root_.scala.actors._
import Actor._
import _root_.javax.mail._
import _root_.javax.mail.internet._

/** Utilities for sending email. */
object Mailer {
  sealed abstract class MailTypes
  sealed abstract class MailBodyType extends MailTypes
  case class PlusImageHolder(name: String, mimeType: String, bytes: Array[Byte])

  case class PlainMailBodyType(text: String) extends MailBodyType
  case class XHTMLMailBodyType(text: NodeSeq) extends MailBodyType
  case class XHTMLPlusImages(text: NodeSeq, items: PlusImageHolder*) extends MailBodyType

  sealed abstract class RoutingType extends MailTypes
  sealed abstract class AddressType(val adr: String) extends RoutingType
  case class From(address: String) extends AddressType(address)
  case class To(address: String) extends AddressType(address)
  case class CC(address: String) extends AddressType(address)
  case class Subject(subject: String) extends RoutingType
  case class BCC(address: String) extends AddressType(address)
  case class ReplyTo(address: String) extends AddressType(address)
```

⁶³ in der Version, wie es aus Lift 1 übernommen wurde

```

implicit def stringToMailBodyType(text: String): MailBodyType = PlainMailBodyType(text)
implicit def xmlToMailBodyType(html: NodeSeq): MailBodyType = XHTMLMailBodyType(html)

case class MessageInfo(from: From, subject: Subject, info: List[MailTypes])

implicit def addressToAddress(in: AddressType): Address = new InternetAddress(in.adr)
implicit def adListToAdArray(in: List[AddressType]): Array[Address] = in.map(a =>
  new InternetAddress(a.adr)).toArray

/** Passwords cannot be accessed via System.getProperty. Instead, we
  * provide a means of explicitly setting the authenticator. */
var authenticator: Box[Authenticator] = Empty

/** The host that should be used to send mail. */
def host = hostFunc()

/** To change the way the host is calculated, set this to the function that calculates
  * the host name. By default: System.getProperty("mail.smtp.host") */
var hostFunc: () => String = _host _

private def _host = System.getProperty("mail.smtp.host") match {
  case null => "localhost"
  case s => s
}

private class MsgSender extends Actor {
  def act = {
    loop {
      react {
        case MessageInfo(from, subject, info) =>
          try {
            val session = authenticator match {
              case Full(a) => Session.getInstance(System.getProperties, a)
              case _ => Session.getInstance(System.getProperties)
            }

            val message = new MimeMessage(session)
            message.setFrom(from)
            message.setRecipients(Message.RecipientType.TO, info.flatMap{ case x: To =>
              Some[To](x) case _ => None})
            message.setRecipients(Message.RecipientType.CC, info.flatMap{ case x: CC =>
              Some[CC](x) case _ => None})
            message.setRecipients(Message.RecipientType.BCC, info.flatMap{ case x: BCC =>
              Some[BCC](x) case _ => None})
            // message.setReplyTo(
            // filter[MailTypes,ReplyTo](info,{case x @ ReplyTo(_) =>
            //   Some(x); case _ => None}))
            message.setReplyTo(info.flatMap{ case x: ReplyTo => Some[ReplyTo](x) case _ =>
              None})
            message.setSubject(subject.subject)
          }
      }
    }
  }
}

```

```

val multiPart = new MimeMultipart("alternative")
info.flatMap{ case x: MailBodyType => Some[MailBodyType](x); case _ =>
  None }.foreach {
  tab =>
    val bp = new MimeBodyPart
    tab match {
      case PlainMailBodyType(txt) => bp.setContent(txt, "text/plain")
      case XHTMLMailBodyType(html) => bp.setContent(html.toString, "text/html")
      case XHTMLPlusImages(html, img @ _*) =>
        val html_mp = new MimeMultipart("related")
        val bp2 = new MimeBodyPart
        bp2.setContent(html.toString, "text/html")
        html_mp.addBodyPart(bp2)
        img.foreach { i =>
          val rel_bpi = new MimeBodyPart
          rel_bpi.setFileName(i.name)
          rel_bpi.setContentID(i.name)
          rel_bpi.setDisposition("inline")
          rel_bpi.setDataHandler(new _root_.javax.activation.DataHandler(
            new _root_.javax.activation.DataSource {
              def getContentType = i.mimeType
              def getInputStream = new _root_.java.io.ByteArrayInputStream(i.bytes)
              def getName = i.name
              def getOutputStream = throw new _root_.java.io.IOException(
                "Unable_to_write_to_item")
            })
          )))
          html_mp.addBodyPart(rel_bpi)
        }
        bp.setContent(html_mp)
      }
    multiPart.addBodyPart(bp)
  }
  message.setContent(multiPart);

  Transport.send(message);
} catch {
  case e: Exception => Log.error("Couldn't_send_mail", e)
}

case _ => Log.warn("Email_Send:_Here..._sorry")
}
}
}

private val msgSender = {
  val ret = new MsgSender
  ret.start
  ret
}

```



```

/** Asynchronously send an email. */
def sendMail(from: From, subject: Subject, rest: MailTypes*) {
  // forward it to an actor so there's no time on this thread spent sending the message
  msgSender ! MessageInfo(from, subject, rest.toList)
}

```

Listing A.17: Objekt Mailer aus dem Paket `net.liftweb.util`

Es folgen die in CometMailer modifizierten Programmzeilen.

```

//Datei src/main/scala/com/lehrkraftnews/util/CometMailer.scala, Ausschnitt imports:
package com.lehrkraftnews.util

import net.liftweb.http.CometActor
import net.liftweb.util._
import net.liftweb.common._

```

Listing A.18: Geänderte Imports in `com.lehrkraftnews.util.CometMailer`

```

//Datei src/main/scala/com/lehrkraftnews/util/CometMailer.scala, Ausschnitt objectPlusLog:
object CometMailer {
  private val log = LoggerFactory.create()
}

```

Listing A.19: Geänderter Objektname und zusätzliches `log` in `CometMailer`

```

//Datei src/main/scala/com/lehrkraftnews/util/CometMailer.scala, Ausschnitt messageTypes:
//Hinzufügung des Parameters "callback" zum Konstruktor der Case-Klasse:
case class MessageInfo(
  from: From, subject: Subject, callback: Box[CometActor], info: List[MailTypes]
)

//Hinzugefügt: case-Klassen für Nachrichten an den callback-CometActor
case class MailSendSuccess(address: String)
case class MailAddressFailure(address: String)
case class MailingFailure(exc: Throwable)

```

Listing A.20: Erweitertes `MessageInfo` und zusätzliche Message-Klassen in `CometMailer`

Im Zweig `case MessageInfo` der `act`-Methode wird auch der zusätzliche Parameter `callback` gemacht. Der einzeilige Methodenaufruf `Transport.send(message)`, der letztendlich die Nachricht versendet, wurde aus Gründen der Effizienz und Diagnosestärke beim Versand einer Nachricht an viele Empfänger so umgestellt, dass dasselbe `Transport`-Objekt für jeden Einzelversand weiter verwendet wird. Außerdem wird bei einer `SendFailedException` eine spezielle Nachricht `MailAddressFailure` an den `Callback` versandt:

```
//Datei src/main/scala/com/lehrkraftnews/util/CometMailer.scala, Ausschnitt send:
{
  //Transport.send(message);
  message.saveChanges(); // implicit with send()
  val transport: Transport = session.getTransport("smtp");
  transport.connect(*/*host, username, password*/); //Check password only once!
  val recipients: Array[javax.mail.Address] = message.getAllRecipients
  val oneRecipient: Array[javax.mail.Address] = new Array[javax.mail.Address](1)
  for(oneAddress <- recipients){
    oneRecipient(0) = oneAddress
    try {
      transport.sendMessage(message, oneRecipient)
      callback.foreach(_ ! MailSendSuccess(oneAddress.toString))
    }
    catch {
      case e: javax.mail.SendFailedException => callback.foreach(c => {
        c ! MailAddressFailure(oneAddress.toString)
        log.error("Couldn't_send_mail_to_address:_", e)
      })
    }
  }
}
}
```

Listing A.21: Ersatz von `Transport.send(message)` in `com.lehrkraftnews.util.CometMailer`

Schließlich wird der Callback-Actor durch folgenden `catch`-Zweig von allgemeinen Fehlern beim Mailversand benachrichtigt:

```
//Datei src/main/scala/com/lehrkraftnews/util/CometMailer.scala, Ausschnitt catch:
} catch {
  case e: Exception => callback.foreach(c => {
    c ! MailingFailure(e);
    log.error("Failure_sending_mails:_", e)
  });
}
```

Listing A.22: Allgemeine Fehlerbehandlung in `com.lehrkraftnews.util.CometMailer`

Auch die für Benutzung von außen vorgesehene Methode `sendMail` wurde um den Parameter `callback` erweitert:

```
//Datei src/main/scala/com/lehrkraftnews/util/CometMailer.scala, Ausschnitt sendMail:
def sendMail(callback: Box[CometActor], from: From, subject: Subject, rest: MailTypes*) {
  // forward it to an actor so there's no time on this thread spent sending the message
  msgSender ! MessageInfo(from, subject, callback, rest.toList)
}
```

Listing A.23: Erweiterung von `sendMail` in `com.lehrkraftnews.util.CometMailer`

Literaturverzeichnis

- [Apac09] Apache Maven Projekt : Maven Features
<http://maven.apache.org/maven-features.html>
- [Apac09a] Apache Wicket : Introduction
<http://wicket.apache.org/introduction.html>
- [ChDW09] Chen-Becker, D., Danciu, M., Weir, T.: The Definitive Guide to Lift
- [Chen09] Chen-Becker, D.: A Brief, But Dense, Intro to Scala <http://www.slideshare.net/dchenbecker/a-brief-but-dense-intro-to-scala>
- [CLBG10] The Computer Language Benchmarks Game on debian.org. Comparison of Scala and Java. <http://shootout.alioth.debian.org/u32/scala.php>
- [Djan10] Django - The Web framework for perfectionists with deadlines
<http://docs.djangoproject.com/en/dev/intro/tutorial02/>
- [Gavi08] InfoQ: David Pollak on lift and Scala
<http://www.infoq.com/news/2008/03/liftweb>
- [GoF94] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software.
http://en.wikipedia.org/wiki/Design_Patterns
- [GoGr09a] Pollack, D.: Lift Google Group: Status of DB backend for record
<http://groups.google.de/group/liftweb/msg/df9cfed48f060215>
- [GoGr09b] Pollack, D.: Lift Google Group: Comet Actors - single browser, multiple tabs, same URL - out of control GET problem
<http://groups.google.de/group/liftweb/msg/da26855ddbf0328>
- [GoGr09c] Pollack, D.: Lift Google Group: Becoming a Scala/Lift Guru
<http://groups.google.de/group/liftweb/msg/983ef90e30c7f8a9>

- [GoGr09d] Indrajit Raychaudhuri: Lift Google Group: Announcing the Lift 2.0 branch
<http://groups.google.de/group/liftweb/msg/6836cca56b3d19ba>
- [GoGr09e] David Pollak: Lift Google Group: Mega*ProtoUser extensibility - a suggestion.
<http://groups.google.de/group/liftweb/msg/a814afedfaaeac36>
- [GoGr09f] David Pollak: Lift Google Group: Shutdown exception
<http://groups.google.de/group/liftweb/msg/31da554e3a39487a>
- [KnHa07] Christoph Knabe, Siamak Haschemi: Ausnahmen bestätigen die Regel. Zentrales Exception Handling mit AspectJ: Der komfortable Notausgang. In Java Magazin 11.07, pp. 23-27, Software & Support Verlag (2007), ISSN 1619-795X http://public.beuth-hochschule.de/~knabe/java/multex/snapshot/JavaMagazin-2007-11p23-27_KnabeHashemi.pdf
- [Lifta] Lift - The Simply Functional Web Framework – Team
<http://liftweb.net/team.html>
- [Liftb] Lift - The Simply Functional Web Framework - Home
<http://liftweb.net/>
- [LiftWikiDBC] Lift Wiki: Mapper - Creating a Database Connection
http://www.assembla.com/wiki/show/liftweb/Mapper#creating_a_database_connection
- [OdSV08] Odersky, M., Spoon, L., Venners, B.: Programming in Scala
- [PhDi09] Phillip Bollman, G., Dietrich, C.: Lift - Vehikel zum nächsten Web-Framework-Level? (Erschienen auf: heise Developer) <http://www.heise.de/developer/artikel/Ajax-und-ORM-787527.html>
- [Poll08] Pollack, D.: Lift View First
http://wiki.liftweb.net/index.php?title=Lift_View_First
- [RuTH08] Ruby, S., Thomas, D., Heinemeier Hansson, D., et al: Agile Web Development with Rails
- [Scal10] Scala Homepage: The Scala 2.8.0 final distribution
<http://www.scala-lang.org/node/7009>
- [ScTo09] Scala Tools: Generating API with ScalaDoc http://scala-tools.org/mvnsites/maven-scala-plugin/usage_doc.html

- [Seas] Seaside - Maintaining State
<http://www.seaside.st/documentation/maintaining-state>
- [TDD] Wikipedia: Testgetriebene Entwicklung
http://de.wikipedia.org/wiki/Testgetriebene_Entwicklung
- [WeMa08] Wetterwald, J., Mao, J.: Project 2: Lift, the Scala Web framework
[http://stanford.wikia.com/wiki/Project_2:
_Lift,_the_Scala_Web_framework](http://stanford.wikia.com/wiki/Project_2:_Lift,_the_Scala_Web_framework)
- [XP] Don Wells: *Extreme Programming: A gentle introduction*
<http://www.extremeprogramming.org/>

Abbildungsverzeichnis

4.1. Startseite des generierten Lift-Projekts mit dem „lift-basic“-Archetyp	15
4.2. Inbetriebnahme der Beispielanwendung über Tomcats „Web Application Manager“	17
4.3. Darstellung der Projektstruktur in Eclipse	18
5.1. Rollenbasiertes Anwendungsfalldiagramm für die Beispielanwendung	21
5.2. MySQL - Datenbankschema der Beispielanwendung	22
5.3. Entity/Relationship-Modell für die Beispielanwendung	22
5.4. Benutzerverwaltung	30
5.5. Bearbeitungsansicht	30
7.1. Fehlermeldung bei gescheiterter Validierung	48
7.2. CRUDify Listenansicht	59
7.3. CRUDify Bearbeitungsansicht	60
8.1. Templates im <i>webapp</i> -Ordner	72
8.2. Fehlermeldung und zugehöriges HTML	83
9.1. Eingebettete Willkommensnachricht	86
10.1. Ansicht: Nachricht hinzufügen	107
11.1. Menü-Gruppen in der Beispielanwendung	132
11.2. Bereich "admin_menu" in der Beispielanwendung	134
11.3. Links „Bearbeiten“ und „Hinzufügen“ in der Ansicht „Eigene Nachricht“ .	136
11.4. Übersicht der Menu-Objekte der Beispielanwendung	138
12.1. Fehlerseite nach dem Provozieren einer Ausnahme mittels Menüpunkt <i>Throw</i>	147
12.2. Fehleranzeige nach dem Provozieren einer AJAX-Ausnahme und Folgenavigation	150
12.3. Fehlerhinweis nach dem Provozieren einer Comet-Ausnahme	153
12.4. Fehlermeldung nach einer Ausnahme in einem Hintergrund-Actor	155
A.1. Paginierung bei der Nachrichtenauflistung	167

Programmauszugsverzeichnis

pom-versions.xml	3
2.1. Scala-Berechnung in einem XML-Ausdruck	7
4.1. Projektübergreifende Maven-Einstellungen	12
4.2. Grundstruktur einer POM-Datei	13
4.3. Befehl zur Erstellung eines Lift-Projekts mit dem lift-basic Archetyp	14
4.4. Verzeichnisstruktur des erstellten Projekts	16
5.1. Umschalten der git-Arbeitskopie auf die zu dieser Buchversion passende <i>Commit-Id</i>	25
5.2. Umschalten der git-Arbeitskopie mit Hilfe der verkürzten <i>Commit-Id</i>	25
5.3. Mail-Zugangskonfiguration in mail.properties	26
5.4. Erzeugen der Beispielbenutzer in der Testsuite	27
5.5. Erzeugen der Beispielabonnements in der Testsuite	27
5.6. Jetty-Plugin mit manuell definiertem Connector für Port 8180 in pom.xml	29
6.1. src/test/scala/platform/ExampleTest.scala	34
6.2. Traditionelles Codebeispiel mit Ausgabe	35
6.3. Das sqrt-Code-Beispiel umformuliert als Testfall	36
7.1. Property-Definitionen in pom.xml	38
7.2. Mapper-Abhängigkeit in pom.xml	38
7.3. Trait net.liftweb.mapper.ConnectionManager	38
7.4. Konfiguration einer H2-Datenbank in der Datei Boot.scala	39
7.5. Einstellung des zentralen Transaktionsmanagements in Datei Boot.scala	40
7.6. Konfiguration des Database Connection Managers in der Datei Boot.scala	40
7.7. Anforderung des H2-Datenbanktreibers in pom.xml	40
7.8. Anforderung des PostgreSQL-Datenbanktreibers in pom.xml	40
7.9. Konfiguration einer PostgreSQL-Datenbank in der Datei Boot.scala	41
7.10. Die Modell-Klasse News	41
7.11. Validierung des Datenfeldes <i>content</i> in der Model-Klasse News	47
7.12. Beispiel: create	48
7.13. Beispiel: verkettetes create	49
7.14. Beispiel: Zugriff auf Datenfelder	49
7.15. Beispiel: Zugriff auf Objekt-Verbindungen	50

7.16. Beispiel: findAll	51
7.17. Beispiel: count	51
7.18. Beispiel: find	51
7.19. Beispiel: By-Parameter	52
7.20. Beispiel: ByList-Parameter mit Typangabe Long	53
7.21. Beispiel: ByList-Parameter mit Long-Literalen	53
7.22. Beispiel: Like-Parameter	53
7.23. Beispiel: NullRef-Parameter	53
7.24. Beispiel: UND-Verknüpfung mehrerer Query-Parameter	54
7.25. Beispiel: Kombination aus Vergleichs- und Steuer-Parametern	54
7.26. Beispiel: Parametriertes SQL-Statement mit dem BySql-Parameter	55
7.27. Beispiel: Unparametriertes SQL-Statement mit dem BySql-Parameter	55
7.28. Beispiel: Prepared SQL-Statement	56
7.29. Ermitteln aller Nachrichten für einen Abonnenten	56
7.30. Konfiguration von Schemifier in bootstrap.liftweb.Boot.boot	58
7.31. Erzeugung von CRUD-Pages für die Modellklasse <i>News</i>	58
7.32. Anpassung des Datenfeldes <i>byUserId</i> für CRUD-Pages der Modellklasse <i>News</i>	59
7.33. Einbettung der News-CRUD-Verwaltung in den Seitenaufbau	61
7.34. Beispiel: Einfache Benutzerklasse mit <i>ProtoUser-Trait</i>	62
7.35. <i>com.lehrkraftnews.model.User</i>	64
7.36. Ausgabe des Vornamens des eingeloggten Benutzers	67
7.37. Funktion <i>loginXhtml</i> aus dem <i>MetaMegaProtoUser-Trait</i>	68
7.38. Definition des „Nachname“-Feldes im <i>ProtoUser-Trait</i>	69
7.39. Festlegen des angezeigten Feldnamens in generierten Formularen	69
7.40. Auflösung der Mehrdeutigkeit beim gleichzeitigen Erben von <i>MetaMegaProtoUser</i> und <i>CRUDify</i>	70
8.1. <i>src/main/webapp/help/howto.html</i>	73
8.2. <i>src/main/webapp/templates-hidden/default.html</i>	73
8.3. <i>src/main/webapp/templates-hidden/main_menu.html</i>	75
8.4. <i>src/main/scala/com/lehrkraftnews/view/Feed.scala</i>	78
8.5. Beispiel: Partielle Funktion in Scala	80
8.6. Partielle <i>dispatch</i> -Funktion in <i>Feed.scala</i>	81
8.7. Einbindung des <i>Msgs</i> -Snippets in das default-Template der Beispielanwendung	82
9.1. <i>src/main/webapp/index.html</i>	85
9.2. <i>com.lehrkraftnews.snippets.HomePage</i>	85
9.3. <i>src/main/webapp/templates-hidden/welcome/welcome_msg.html</i>	86
9.4. Auszug aus <i>/src/main/webapp/news/viewSingle.html</i>	87
9.5. <i>com.lehrkraftnews.snippet.NewsSnippet.showSingle</i>	88
9.6. <i>webapp/templates-hidden/tableTemplates/_byTeacherNewsTable.html</i>	89
9.7. <i>com.lehrkraftnews.snippet.NewsSnippet.showByTeacher</i>	90
9.8. Anzeige aller News in <i>/src/main/webapp/news/index.html</i>	91

9.9. com.lehrkraftnews.snippet.NewsSnippet.showAll	91
9.10. Beispiel: Template für ein einfaches Formular	92
9.11. Beispiel: Einfaches Formular-Snippet	92
9.12. Beispiel: Erzeugter HTML-Code	93
9.13. Beispiel: Einfaches Formular mit RequestVar	94
9.14. Beispiel: Einfaches Formular mit RequestVar und link	95
9.15. Beispiel: Template für Snippet in Listing 9.14	95
9.16. Beispiel: Generierter Link	95
9.17. com.lehrkraftnews.snippets.NewsSnippet.newsToDelete	95
9.18. Löschen einer Nachricht in src/main/webapp/news/delete.html	96
9.19. com.lehrkraftnews.snippet.NewsSnippet.deleteNews	96
9.20. src/main/webapp/subscriptions/manage.html	98
9.21. com.lehrkraftnews.snippet.SubscriptionSnippet	98
9.22. Generiertes Formular (mit Beispieldaten) aus der Abonnement-Verwaltung	101
10.1. Auszug aus src/main/webapp/news/viewByTeacher.html	104
10.2. webapp/templates-hidden/tableTemplates/_byTeacherNewsTable.html . . .	104
10.3. com.lehrkraftnews.snippet.NewsSnippet.teacherSelect	104
10.4. Beispiel: Zusätzliches Closure durch hidden-Element in AJAX-Formularen	105
10.5. Comet-Formular in src/main/webapp/news/edit.html	108
10.6. Rückmeldungsbereich in src/main/webapp/news/edit.html	108
10.7. MailComet: Definition des SessionVar-Objekts	109
10.8. MailComet: Hilfsvariablen	110
10.9. MailComet: render-Methode	110
10.10. Verwendung der Helpers.bind-Methode innerhalb von CometActor-Klassen	111
10.11. MailComet: doSave-Methode	111
10.12. MailComet: sendNewsMail-Methode	112
10.13. MailComet: lowPriority-Methode	113
10.14. MailComet: case object für Reset-Message	114
10.15. MailComet: updateSendStatus-Methode	115
10.16. Einbindung des CometActors NewsReader im default-Template	116
10.17. com.lehrkraftnews.comet.NewsReader	116
10.18. com.lehrkraftnews.comet.NewsMaster	118
11.1. LiftRules-Konfiguration in bootstrap.liftweb.Boot.boot	121
11.2. Mailer-Konfiguration in bootstrap.liftweb.Boot.boot	123
11.3. Objekt zum Einlesen der Mail-Properties	124
11.4. Beispiel: Erstellung einer einfachen SiteMap-Struktur	125
11.5. Beispiel: Zugriffssteuerung durch If-Objekte	126
11.6. Definition der If-Objekte aus der Beispielanwendung in Boot.boot	127
11.7. Beispiel: SiteMap-Definition in der Lift-Live-Demo-Applikation	128
11.8. Vorbereitung zur Menüdefinition in bootstrap.liftweb.Boot.boot	129
11.9. Menüdefinition in bootstrap.liftweb.Boot.boot	130

11.10	Beispiel: Menü mit <code>Menu.builder</code>	131
11.11	Beispiel: Generierte HTML-Liste für das Beispiel aus Listing 11.5	131
11.12	Beispiel: Ausschluss eines Ortes aus dem Menü durch <code>Hidden</code>	132
11.13	Definition der Menügruppe „main“ aus der Beispielanwendung	132
11.14	Erzeugung von Gruppenmenüs im Template <code>/templates-hidden/main_menu</code>	133
11.15	Hinzufügen von Menüpunkten aus <code>User.siteMap</code> zur Menügruppe „user“	133
11.16	Einbindung einzelner Menüpunkte in <code>/templates-hidden/admin_menu</code>	134
11.17	Definition des Admin-Menüs in <code>Boot.boot</code>	134
11.18	Erstellung dynamischer Seiten-Titel mit <code>Menu.title</code> im <code>default</code> -Template	135
11.19	Erstellung der Aktionen-Links in der Methode <code>NewsSnippet.showOwn</code>	135
11.20	Definition des <code>Menu</code> -Objekts für die Bearbeitung von Nachrichten	136
11.21	Erstellung eines <code>Template</code> -Objekts	137
11.22	Definition des <code>Menu</code> -Objekts für die Erstellung neuer Nachrichten	137
11.23	Definition des <code>Menu</code> -Objekts für die Detailansicht von Nachrichten	138
11.24	Umleitung aller Requests auf den Pfad <code>/news/viewSingle</code> (dadurch Endlosschleife)	139
11.25	Definition einer <code>URL-Rewrite-Funktion</code> in <code>Boot.boot</code>	139
12.1.	<code>src/main/webapp/throw.html</code>	142
12.2.	<code>com.lehrkraftnews.snippet.NewsSnippet.throwException</code>	142
12.3.	Eine durch <code>Lift</code> angezeigte Standard-Fehlerseite	142
12.4.	Signatur der Ausnahmebehandler von <code>Lift</code>	143
12.5.	Einsetzen eines primitiven <code>ExceptionHandler</code> in der Klasse <code>Boot</code>	143
12.6.	Der Standard-Ausnahmebehandler von <code>Lift</code>	143
12.7.	Einsetzen eines zentralen <code>ExceptionHandler</code> in der Klasse <code>Boot</code>	144
12.8.	Aufbau einer Fehlerseite in der Klasse <code>ExceptionResponse</code>	145
12.9.	Methode <code>apply</code> des Objekts <code>ExceptionReporting</code>	148
12.10	Methode <code>exceptionHandler</code> der Klasse <code>DiagnosticCometActor</code>	151
12.11	Ausnahmebehandlung im Hintergrund- <code>Actor CometMailer</code>	154
13.1.	Definition des Menüpunktes „Ausloggen“ innerhalb des <code>MetaMegaProtoUser-Traits</code>	157
13.2.	Ausschnitt aus der Datei <code>lift-core_de.properties</code> für deutsche Sprache	158
13.3.	Zugriff auf internationalisierte Zeichenketten innerhalb von <code>Templates</code>	159
A.1.	Anforderung von Logging mittels <code>SLF4J</code> über <code>Log4J</code> in <code>pom.xml</code>	163
A.2.	Konfiguration des Loggings mittels <code>log4j</code>	164
A.3.	Explizite Initialisierung von <code>Log4J</code>	164
A.4.	Beispiel: <code>Box</code> und <code>foreach</code>	165
A.5.	Beispiel: <code>Box</code> und <code>map</code>	165
A.6.	Beispiel: <code>Box</code> und <code>open!</code>	166
A.7.	Beispiel: <code>Box</code> und <code>openOr</code>	166
A.8.	Das <code>Singleton-Objekt Paginator</code>	166

A.9. Nutzung des Paginator-Objekts im showAll-Snippet	168
A.10. Widgets-Abhängigkeit in pom.xml	169
A.11. Verwendung des TableSorter-Widgets im Template news/index.html . . .	169
A.12. com.lehrkraftnews.snippet.TableSorterNewsSnippet	170
A.13. Einfache Actors	172
A.14. Versand von Messages an einen Actor	173
A.15. case-Objekte als Message	174
A.16. case-Klassen als Message	174
A.17. Objekt Mailer aus dem Paket net.liftweb.util	176
A.18. Geänderte Imports in com.lehrkraftnews.util.CometMailer	179
A.19. Geänderter Objektname und zusätzliches log in CometMailer	179
A.20. Erweitertes MessageInfo und zusätzliche Message-Klassen in CometMailer	179
A.21. Ersatz von Transport.send(message) in com.lehrkraftnews.util.Comet- Mailer	180
A.22. Allgemeine Fehlerbehandlung in com.lehrkraftnews.util.CometMailer . . .	180
A.23. Erweiterung von sendMail in com.lehrkraftnews.util.CometMailer	180