Diplomarbeit

Umsetzung einer verteilten Anwendung mit der dokumentenorientierten Datenbank CouchDB

Ein Gliederungseditor als replizierbares Verteiltes System

Lena Herrmann

Beuth Hochschule für Technik Berlin
University of Applied Sciences

Upstream-Agile GmbH

Fachbereich VI Informatik und Medien
Studiengang Medieninformatik, Schwerpunkt Software
Matrikelnummer 720742

Betreuer: Prof. Dr. Stefan Edlich
Gutachter: Prof. Dr. Frank Steyer

Eingereicht am 14. Juli 2010

«For most of mankind's history we have lived in very small communities in which we knew everybody and everybody knew us. But gradually [...] our communities became too large and disparate [...], and our technologies were unequal to the task of drawing us together. But that is changing.
Interactivity. Many-to-many communications. Pervasive networking. These are cumbersome new terms for elements in our lives so fundamental that, before we lost them, we didn't even know to have names for them.»

(Douglas Adams, 1999)

Abstract

Auf heutigen Webbrowsern und mobilen Endgeräten können komplexe Anwendungen ausgeführt werden. Diese ermöglichen Zusammenarbeit und Datenaustausch zwischen BenutzerInnen. Bei Laptops und Mobilfunkgeräten kann keine kontinuierliche Internetverbindung vorausgesetzt werden. Um dieses Problem zu berücksichtigen kann Datenreplikation eingesetzt werden, wobei die Daten regelmäßig synchronisiert und dabei konsistent gehalten werden müssen. In dieser Arbeit wird die Konzeption und prototypische Erstellung einer JavaScript-Anwendung beschrieben, die mithilfe der dokumentenorientierten Datenbank CouchDB einen verteilten Gliederungseditor umsetzt. Mit einem Gliederungseditor können z. B. Gedanken oder Konzepte hierarchisch geordnet aufgeschrieben werden. Neben der Einordnung des zu erstellenden Systems und einer Analyse der in Frage kommenden Lösungsmöglichkeiten werden die verwendeten Technologien beschrieben. Genau vorgestellt werden dabei CouchDB mit der eingebauten Master-Master-Replikation und der Möglichkeit, eine komplexe Applikation ohne Middleware zu implementieren. Die erstellte Anwendung läuft lokal im Browser und ist dadurch auch offline benutzbar. Konflikte werden bei der Synchronisation vom System, teilweise mit Benutzerunterstützung, aufgelöst. In der Arbeit wird abschließend die Einsetzbarkeit von CouchDB für eine verteilte Anwendung und speziell für den gewählten Anwendungsfall beurteilt.

Gliederung

1 Einleitung

1.1 Motivation

> We're at the dawn of a new age on the web, or maybe on a horizon, or a precipice; the metaphor doesn't matter, the point is, things are changing. Specifically, as browsers become more and more powerful, as developers we're given new opportunities to rethink how we architect web applications. [Qui09]

Das Internet wird mehr und mehr ein Bestandteil des alltäglichen Lebens. Der Anteil der regelmäßigen Internetnutzer in Deutschland liegt im Jahr 2010 bei 72 Prozent und steigt stetig an. Unter den 14 bis 19jährigen beträgt der Anteil der Menschen, die das Internet nie nutzen, nur noch 3 Prozent [Ini10, S. 10]. Mit der Anzahl der Benutzer nimmt die Normalität zu, mit der Webanwendungen für Zusammenarbeit, Kommunikation und Datenaustausch benutzt werden. Dabei müssen diese Webanwendungen einer steigenden Anzahl von Benutzern zeitgleichen Zugriff erlauben. Dem Webbrowser kommt dabei als Plattform für Anwendungen eine immer größere Bedeutung zu [Pau05, S. 16)]. Kaum eine Software hat sich in den letzten zehn Jahren so weiterentwickelt wie der Webbrowser [GJ05]. Dadurch entstehen auch neue Möglichkeiten, Webanwendungen zu entwickeln: Es können immer höhere Anforderungen an Bedienbarkeit, Einsetzbarkeit und Verfügbarkeit der Anwendungen gestellt und auch erfüllt werden.

Ein weiterer Trend ist die zunehmende Verbreitung von mobilen Endgeräten [Ini10, S. 61]. In [Kot99, S. 7] wird vorausgesagt, dass durch das Wachstum der Angebote im Internet in Verbindung mit der Weiterentwicklung der Informationstechnologien die Anzahl der Zugangsgeräte stark ansteigen wird. Mobile Endgeräte sind heute vor allem Laptops und Mobiltelefone. Diese haben oft eine schlechtere Konnektivität als stationäre Endgeräte. Lange Phasen ohne Verbindung sind die Norm, weitere Herausforderungen sind hohe Latenzen und limitierte Bandbreite [Guy98].

Es besteht also ein großer Bedarf an Technologien, die die oben aufgeführten Ansprüche an Zusammenarbeit und Datenaustausch erfüllen. Diese müssen die Umsetzung von Systemen ermöglichen, die in großem Rahmen skalieren. Gleichzeitig sollen sie aber auch den Bedarf an kontinuierlicher Konnektivität stark reduzieren. Eine mögliche Lösung für diese Art Problem ist Datenreplikation. Diese wird bei [Guy98] so umrissen:

> Copies of data are placed at various hosts in the overall network, [...] often local to users. In the extreme, a data replica is stored on each mobile computer that desires to access that data, so all user data access is local.

Die Schwierigkeit hierbei ist, die Daten regelmäßig zu synchronisieren und konsistent zu halten. Die Überwachung der Konsistenz ist Aufgabe des Systems, das die Replikation durchführt. Ziele eines solchen Systems sind hohe Verfügbarkeit und Kontrolle über die eigenen Daten. Eine solche Lösung kann gleichzeitig den hohen Ansprüchen an Datenschutz genügen, die Benutzer an moderne Webanwendungen stellen [Kra09a, Kra09b]. Bei Systemen, die Replikation umsetzen, können die Benutzer selbst entscheiden, mit wem sie ihre Daten teilen. Bei entsprechender Implementierung des Systems ist die Benutzung außerdem zumindest zeitweise unabhängig von zentralen Servern. Der Ausfall des Netzwerks oder einzelner Knoten ist von vornherein mit eingeplant.

In dieser Arbeit wird die Konzeption und Umsetzung einer Software beschrieben, die die Datenbank CouchDB verwendet. Mit CouchDB können Anwendungen umgesetzt werden, die sowohl „nach oben“ als auch „nach unten“ skalieren: Anwendungen sollen sich auf beliebig viele Knoten verteilen lassen, um Verfügbarkeit und *Performance (Leistung)* zu gewährleisten. Sie sollen aber auch auf mobilen Endgeräten einsetzbar sein und Synchronisierung von Benutzerdaten ermöglichen [Leh10].

1.2 Aufbau der Arbeit

Zu Beginn wird die zentrale Fragestellung der Arbeit definiert (Kapitel 2). Das Ziel dieser Arbeit ist die fundierte Beantwortung dieser Frage. Um dies zu ermöglichen, wird in Kapitel 3 zunächst eine Einordnung des zu erstellenden Systems und eine Analyse der in Frage kommenden Lösungsmöglichkeiten vorgenommen.

Kapitel 4 widmet sich der Datenbank CouchDB. In Abschnitt 4.1 wird diese theoretisch eingeordnet, in Abschnitt 4.2 werden die Umsetzungsdetails erklärt. In Kapitel 5 werden weitere Technologien vorgestellt, die in die Umsetzung der Anwendung eingeflossen sind. Beschrieben werden nacheinander die verwendeten Webtechnologien (Abschnitt 5.1), Cloud Computing (Abschnitt 5.2) sowie die unterstützenden Werkzeuge (Abschnitt 5.3).

Die Anforderungen an die Anwendung werden in Kapitel 6 spezifiziert. In Kapitel 7 wird die Struktur der Anwendung umrissen, auch hierbei werden verschiedene Designalternativen gegeneinander abgewogen. Die technischen Details des fertigen Systems sind in Kapitel 8 skizziert, ergänzt von Quelltextauszügen im Anhang. Eine Bedienungsanleitung für die Anwendung in Kapitel 9 schließt den praktischen Teil der Arbeit ab. Eine Beurteilung des Ergebnisses der Arbeit wird abschließend in Kapitel 10 vorgenommen.

1.3 Textauszeichnung

Die Informatik ist ein Gebiet, in dem sich viele englische Begriffe im Deutschen eingebürgert haben. Für viele dieser englischen Begriffe gibt es noch keine Entsprechung, für manche werden in der Literatur unterschiedliche deutsche Übersetzungen verwendet. In dieser Arbeit wird dort, wo der englische Begriff im Deutschen üblicherweise verwendet wird, der englische Originalbegriff beibehalten. Eine Neuübersetzung von englischen Begriffen wurde bewusst vermieden. Eine deutsche Übersetzung ist für bessere Verständlichkeit beim ersten Auftreten des Begriffs in Klammern angegeben. Gibt es im Deutschen einen passenden Begriff, wird die englische Bedeutung beim ersten Auftreten des Begriffs in Klammern angegeben.

Zur besseren Übersichtlichkeit und Lesbarkeit dieser Arbeit werden manche Begriffe besonders hervorgehoben. Im Text erklärte Fachbegriffe sowie Namen von eingesetzten Technologien werden bei ihrem ersten Auftreten *kursiv* gesetzt. Bei erneutem Auftreten werden solche Begriffe nicht besonders ausgezeichnet. Im Abkürzungsverzeichnis im Anhang A.1 können verwendete Abkürzungen nachgeschlagen werden. Begriffe aus dem Quelltext werden bei jedem Vorkommen durch die Verwendung der Schriftart `Courier` gekennzeichnet. Quelltextauszüge sind in einer `Typewriter`-Schrift gesetzt, als Block gesetzte Quelltextauszüge sind außerdem hell hinterlegt und mit einem Rahmen versehen. Zitatblöcke sind rechts und links eingerückt. Zitate im Fließtext sind in „Apostrophe“ gesetzt.

2 Aufgabenstellung

In der Einleitung wurde auf die veränderten Gegebenheiten und erhöhten Ansprüche an Anwendungsprogramme in der heutigen Zeit (Juni 2010) hingewiesen. Mehr Menschen nutzen gleichzeitig mit immer mehr mobilen Endgeräten immer größere Systeme und stellen dabei immer höhere Ansprüche an Benutzbarkeit und Verfügbarkeit.

Gleichzeitig schreitet die Entwicklung von Technologien voran, mit der diese gestiegenen Anforderungen immer besser umgesetzt werden können. Im Bereich der Datenbanksysteme entwickelte sich im letzten Jahrzehnt eine „Bewegung" mit dem Namen *NoSQL* [Str98]. NoSQL ist eine nicht näher eingegrenzte Bezeichnung für eine Reihe nichtrelationaler Datenbanksysteme bzw. *Key-Value-Stores*. Sie haben gemeinsam, dass sie im Gegensatz zu relationalen Datenbanken meist keine festen Tabellenschemata benötigen. Besonderen Schwerpunkt setzen sie auf Verteilbarkeit und eignen sich dadurch meist für die Skalierung im großen Maßstab. Die Vorzüge traditioneller Datenbanksysteme, insbesondere die Konsistenz zu jedem Zeitpunkt, werden oft gegen eine bessere Verfügbarkeit oder Partitionstoleranz (s. Abschnitt 4.1.2) getauscht.

Diese neuen Datenbanksysteme wurden jeweils für spezifische Anwendungsfälle entwickelt. Ein umfassender Vergleich von NoSQL-Datenbanksystemen ist nicht Gegenstand dieser Arbeit. Stattdessen soll die Einsetzbarkeit eines bestimmten NoSQL-Datenbanksystems anhand eines konkreten Anwendungsfalles überprüft werden. Kann nach eingehender Analyse mit der ausgewählten Technologie eine funktionsfähige Software umgesetzt werden?

Die dokumentenorientierte Datenbank CouchDB [Apa09c] wurde in der Einleitung bereits kurz vorgestellt. Bei einer mit CouchDB implementierten Anwendung kann der Einsatz von *Middleware* völlig entfallen. *Master-Master-Replikation* ist eine Kernfunktion, was CouchDB besonders geeignet für verteiltes Arbeiten macht. CouchDB-Anwendungen laufen direkt aus dem Webbrowser heraus, deshalb ist die Anzahl der für den Betrieb zu installierenden Programme minimal. So kann das System auf einer möglichst hohen Anzahl von Endgeräten eingesetzt werden. Die Wahl gerade dieses Datenbanksystems wird in den Kapiteln 3 und 4 genauer begründet.

Als Anwendungsfall für den Einsatz von CouchDB wurde ein *Outliner (Gliederungseditor)* gewählt. Mit einem Outliner können z. B. Gedanken oder Konzepte hierarchisch geordnet aufgeschrieben werden. Als Vorlage dient das Programm OmniOutliner [Omn10], ein lokales Desktop-Programm für Mac OS X. Ziel ist, einen Gliederungseditor mit ähnlicher, aber eingeschränkter Funktionalität prototypisch zu erstellen und dadurch die Einsetzbarkeit von CouchDB zu analysieren.

Die hier konzipierte Anwendung unterscheidet sich in einem zentralen Punkt von der Vorlage: Mit ihr soll gemeinschaftliches Arbeiten an Dokumenten auch verteilt über Netzwerke ermög-

licht werden, selbst wenn der Benutzer zwischenzeitlich vom Internet getrennt ist. Die für die Arbeit erstellte Anwendung wird lokal im Browser laufen und offline benutzbar sein. Über eine Internetverbindung werden Daten von mehreren Benutzern gleichzeitig bearbeitet werden können.

In dieser Diplomarbeit soll also untersucht werden, ob sich CouchDB dafür eignet, verteilte Anwendungen zu erstellen. Um dies zu prüfen, wird der Prototyp eines verteilten Gliederungseditors entworfen und umgesetzt.

3 Analyse

In Kapitel 2 wurden Anforderungen an ein System grob skizziert. Entwurf und Implementierung dieser Anwendung sowie die Auswertung des Ergebnisses sind Gegenstand dieser Arbeit. Im folgenden Kapitel werden die Problemstellungen, die die Anwendung lösen soll, sowie die wissenschaftliche Einordnung der Anwendung noch einmal näher untersucht. Dann werden unterschiedliche Lösungsansätze daraufhin analysiert, mit welchen Mitteln ein solches System am besten umzusetzen ist. Dabei wird auf alternative Ansätze eingegangen.

3.1 Anforderungen an einen Gliederungseditor

Bevor der hier gewählte Lösungsweg von anderen Möglichkeiten abgegrenzt wird, werden zunächst die Eigenschaften und Anforderungen der umzusetzenden Anwendung genauer festgelegt.

3.1.1 Definition

Ein Gliederungseditor oder Outliner wird bei Wikipedia als eine Mischung aus einer Freiform-Datenbank und einem Texteditor beschrieben [Wik10a]:

> An outliner is a computer program that allows one to organize text into discrete sections that are related in a tree structure or hierarchy. Text may be collapsed into a node, or expanded and edited. [Wik10c]

Manche Gliederungseditoren ermöglichen außerdem eine Formatierung der Einträge und das Einbinden von verschiedenen Medientypen. Es gibt eine Vielzahl von Implementierungen dieser Art Programm.

Eine der am höchsten bewerteten Umsetzungen [Mac05] ist die kommerzielle Software OmniOutliner [Omn10] (Screenshot s. Abb. A.2). Dieses Programm wird von der Firma „Omni Group“ für das Betriebssystem Mac OS X entwickelt. Es hat neben den üblichen Eigenschaften eines Gliederungseditors noch weitere spezielle Features. OmniOutliner soll beim Entwurf der Anwendung als Vorlage dienen, wenn auch in diesem Prototypen längst nicht alle seine Merkmale umgesetzt werden können.

3.1.2 Einsatzmöglichkeiten

Die Einsatzmöglichkeiten eines Outliners sind sehr vielfältig, ähnlich wie beispielsweise die eines Texteditors. Im Folgenden werden nur einige der möglichen Szenarien für die Benutzung eines Outliners vorgestellt. Einige Szenarien sind an [Wik10a] angelehnt, einige wurden von der Autorin selbst festgelegt. Den Entwurfsentscheidungen bei der Softwareentwicklung liegen diese Anwendungsfälle zugrunde.

Der ursprüngliche Zweck eines Gliederungseditors wird in [Wik10a] als der von Geisteswissenschaftlern häufig verwendete Zettelkasten beschrieben:

> Mehr oder weniger große Textabschnitte (z. B. Zitate) werden in einer nach Kategorien sortierten und verschlagworteten Datei abgelegt und stehen so zur Verfügung. [...] Auf diese Weise lässt sich schnell ein umfangreiches Archiv an wichtigen Textpassagen aufbauen.

Ein häufig zitierter Anwendungsfall für Gliederungseditoren ist das Verfassen von literarischen, journalistischen oder wissenschaftlichen Texten. Zuerst kann der Umriss der Handlung strukturiert nach Kapiteln und Szenen eingegeben werden. Anschließend kann der Benutzer Details in die Äste des Gliederungsbaums einpflegen. Später können Teile des Textes verschoben werden, in dem man die Knoten in der Baumstruktur hin- und herbewegt. Dies ist bei einem Texteditor oder auch einem Textverarbeitungsprogramm wesentlich schwieriger umzusetzen.

Gliederungseditoren können auch immer dann eingesetzt werden, wenn eine größere Menge kürzerer Textpassagen gespeichert werden sollen. Dies kann z. B. Aufgaben, Ideen, Protokolle, Tagebücher oder Einkaufslisten umfassen.

3.2 Anforderungen an Verteilte Systeme

In diesem Abschnitt wird zunächst eine Definition von Verteilten Systemen vorgenommen. Es wird gezeigt, dass es sich bei der zu erstellenden Anwendung um ein Verteiltes System handelt. In den darauffolgenden Abschnitten wird dann begründet, warum CouchDB für die Umsetzung eines solchen Systems gewählt wurde.

Eine vielzitierte Definition findet sich in [Tan07, Kap. 1.1]:

> Ein Verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen.

Diese Beschreibung kann sich auf eine beliebig große Anzahl von Rechnern beziehen. So werden in [Cou05, Kap. 1.2] und in [Ben04, S. 6] auch das Internet bzw. das World Wide Web als Verteilte

Systeme bezeichnet. Es existieren aber auch weitergefasste Definitionen, in denen nicht nur die pysikalisch verteilte Hardware, sondern auch logisch verteilte Anwendungen als Verteiltes System bezeichnet werden [Ben04, S. 5]. Nach [Cou05, Kap. 1.1] ist es die grundlegende Eigenschaft eines Verteilten Systems, ausschließlich über *Message Passing* zu kommunizieren. Allgemein kann gesagt werden:

> A distributed system is a system which operates robustly over a wide network. [And10, Kap. 2]

Ein System ist demnach ein Verteiltes System, wenn es in mehrere Komponenten getrennt werden kann, die autonom sind - das heißt, sie müssen jeweils für sich wieder ein vollständiges System bilden. Die einzelnen Subsysteme können hierbei sehr inhomogen sein, was Implementierung bzw. Betriebssysteme und Hardware angeht. Die Art der Verbindung kann ebenfalls beliebig umgesetzt sein. Außerdem sollte für Benutzer wie für Programme, die das System benutzen, der Eindruck entstehen, es mit einem einzigen System zu tun zu haben. Bei der Entwicklung Verteilter Systeme stellt sich die Aufgabe festzulegen, wie die Komponenten des Systems transparent für den Benutzer zusammenarbeiten. Der innere Aufbau des Systems soll für den Benutzer nicht ersichtlich sein. Weiterhin soll mit dem System konsistent und einheitlich gearbeitet werden können, unabhängig von Zeitpunkt und Ort des Zugriffs [Tan07, Kap. 1.1].

Nach [Cou05, Kap. 1.1] und [Tan07, Kap. 1.1] müssen beim Entwurf von Verteilten Systemen folgende Herausforderungen beachtet werden:

Nebenläufigkeit (Concurrency): Komponenten eines Systems führen Programmcode nebenläufig aus, wenn sie auf gemeinsam genutzte Ressourcen gleichzeitig lesend und schreibend zugreifen, sich aber gegenseitig nicht beeinflussen. Bei einem Verteilten System müssen daher auf nebenläufige Prozesse ausgerichtete Algorithmen verwendet werden.

Keine globale Uhr: Die Uhren der Subsysteme lassen sich nicht synchron halten. Für Auflösung von Bearbeitungskonflikten können daher keine Zeitstempel verwendet werden. *MVCC (Multi Version Concurrency Control)* verwendet stattdessen oft *Vektoruhren* [Lam78] oder andere eindeutige Identifikationsmechanismen wie *UUIDs (Universally Unique Identifier)* [Lea05].

Ausfall von Subsystemen: Der Ausfall einer einzigen Komponente oder eine Störung im Netzwerk dürfen das Gesamtsystem nicht beeinträchtigen. Die Subsysteme müssen unabhängig voneinander funktionieren, auch wenn die Verbindung zwischen ihnen zeitweise oder länger abbricht, oder eines von ihnen nicht mehr funktioniert.

In Kapitel 4.1 wird auf diese Probleme im Detail eingegangen.

3.3 Anforderungen an Groupware

Groupware-Systeme sind nach [Ell89] ...

> ... computer-based systems that support two or more users engaged in a common task, and that provide an interface to a shared environment. These systems frequently require fine-granularity sharing of data and fast response times.

Wie bei jedem Software-Projekt dürfen beim Entwurf eines Verteilten Systems nicht nur technische Möglichkeiten eine Rolle spielen. Ebenso müssen die realen Anforderungen der angestrebten Benutzergruppen bei der Planung miteinbezogen werden. So soll hier der Frage nachgegangen werden, was die spezifischen Fragen und Probleme bei der Benutzung einer Groupware sind, und was demzufolge bei Entwurf, Entwicklung und Schulung besonders zu beachten ist.

Die in dieser Arbeit eingesetzte Datenbank CouchDB wird oft mit Lotus Notes verglichen. Damien Katz, der Erfinder von CouchDB, ist ehemaliger Notes-Entwickler und war nach eigenen Aussagen bei der Entwicklung von CouchDB von Notes stark inspiriert [Sch08]. Notes ist CouchDB insofern ähnlich, als dass eine Datenbank in Notes eine Sammlung von halbstrukturierten Dokumenten ist, die durch Views organisiert sind [Kaw88, Kap. 2]. Nach [Kaw88] ist Lotus Notes ein Kommunikationssystem, das es Gruppen erlaubt, unterschiedliche Informationen wie Texte und Notizen zu teilen und gemeinsam zu bearbeiten:

> The system supports groups of people working on shared sets of documents and is intended for use in a personal computer network environment in which the database servers are „rarely connected“. [Kaw88, Kap. 1]

Die Parallelen zwischen Lotus Notes und der hier entwickelten Anwendung liegen also darin, dass letztere zumindest gleiche Teilaufgaben von Notes erfüllen soll. Außerdem erfolgt die Synchronisierung der Dokumente ebenfalls durch Datenbank-Replikation. Demnach sind wissenschaftliche Erkenntnisse über die Benutzung von Lotus Notes für die Konzeption der Anwendung durchaus interessant.

Mehrere Studien haben die Auswirkungen der Einführung von Lotus Notes auf die Zusammenarbeit in verteilten Gruppen untersucht.

In [Van96] wurde der Einfluss von Notes auf die Zusammenarbeit in einer großen Versicherungsgesellschaft analysiert. Obwohl die Mitarbeiter mit dem Produkt sehr zufrieden waren, wurde keine verstärktere oder bessere Zusammenarbeit unter ihnen festgestellt. Die Studie kommt zu dem Schluss, dass ein System sehr genau zu der Zielgruppe passen muss, und dass der gründlichen Schulung in dieser neuen Technologie eine zentrale Bedeutung beikommt.

Die Autorin von [Kar95] stellt fest, dass Benutzer, die noch nie vorher mit einer Groupware gearbeitet haben, das neue Programm meist wie ein ihnen vertrautes (also lokales Single-User-)

Programm benutzen:

> [The] findings suggest that when people neither understand nor appreciate the co-operative nature of groupware, it will be interpreted as an instance of some more familiar technology and used accordingly. This can result in counter-productive and uncooperative practice and low levels of use. [Kar95, S. 4]

Dies wird von einer anderen Untersuchung bestätigt:

> The findings suggest that where people's mental models do not understand or appreciate the collaborative nature of groupware, such technologies will be intepreted and used as if they were more familiar technologies, such as personal, stand-alone software (e.g., a spreadsheet or word processing program). [Orl92, S. 1]

Wenn die Konstruktion der Software von der Organisationskultur der Gruppe abweicht, wird die Software mit hoher Wahrscheinlichkeit nicht dazu beitragen, sinnvoll kollektiv genutzt zu werden. Eine Groupware muss vielmehr auf die bestehenden Arbeitsabläufe innerhalb einer Gruppe angepasst werden, um Arbeitsprozesse zu verbessern. Umfasst die Aufgabe der Software Konfliktbearbeitung, ist es für einen Erfolg der Software ebenfalls wichtig, dass diese die üblichen Konfliktlösungsstrategien des Teams unterstützt. [Mono1]

Es bleibt festzustellen, dass eine Software, mit deren zentralen Charakteristika die Benutzer noch nicht vertraut sind, zum einen genau an die Zielgruppe angepasst werden muss. Dies ist bei dieser Aufgabenstellung schwierig, da der Prototyp für die geplante Anwendung für keine genau abgegrenzte Zielgruppe entwickelt wird. Die Aufgabe der Eingrenzung der Zielgruppe verbleibt für eine spätere Entwicklungsphase. Zum anderen muss der Schulung bzw. der Dokumentation für die Benutzer eine hohe Bedeutung zu kommen. Die hier entwickelte Arbeit soll einen Beitrag dazu leisten, Menschen an die verstärkte Kooperation mithilfe von Software zu gewöhnen.

3.4 Verschiedene Lösungsansätze

Im Folgenden werden unterschiedliche Lösungsansätze für die Bearbeitung der Aufgabenstellung vorgestellt. Vor- und Nachteile bestehender Lösungen werden diskutiert und die angestrebte Alternative wird herausgehoben.

Das zu lösende Problem ist: Wie können Dokumente von mehreren Benutzern gemeinsam bearbeitet werden, auch wenn manche von ihnen für längere Zeit vom Internet getrennt sind? Wie kann das System auftretende Bearbeitungskonflikte möglichst automatisch behandeln oder sie in einer für den Benutzer geeigneten Form zur Lösung aufbereiten?

3.4.1 Manueller Austausch von Dokumenten

Das trivialste Verfahren ist das manuelle Synchronisieren. Dabei werden Dokumente direkt zwischen den einzelnen Benutzern ausgetauscht, z. B. per Email oder FTP, und nebenläufig bearbeitet. Verschiedene Versionen müssen von einem Benutzer umständlich per Hand zusammengeführt werden, das Resultat dessen muss selbst wieder ausgetauscht werden. Details können hierbei leicht verlorengehen.

Manche Webdienste stellen Netzwerkdateisysteme bereit, mit dem das Synchronisieren der Daten automatisch erledigt werden kann. Der Anbieter Dropbox [Dro10] beispielsweise erlaubt es, Verzeichnisse und Dateien zwischen verschiedenen Rechnern auf einem Stand zu halten. Tritt zwischenzeitlich ein Konflikt auf, werden die verschiedenen Revisionen als einzelne Dateien abgelegt. Es ist dann wieder Sache des Benutzers, die Konflikte aufzulösen.

3.4.2 Echtzeit-Texteditoren

Ein anderer Ansatz, der zunehmend Verbreitung findet, sind zentralisierte Kooperationssysteme über das Internet. Benutzer können mit solchen Webanwendungen meist in Echtzeit gemeinsam an Dokumenten arbeiten. Beispiele sind Etherpad [Fou10], Google Docs [Bel10] und das neuere Google Wave [Goo10]. In letzterem steht der Kommunikationscharakter im Vordergrund, doch können auch hiermit längere Texte gleichzeitig bearbeitet werden. Google Docs hat im Vergleich zu Google Wave mehr die Eigenschaften eines Textverarbeitungsprogramms.

Desktop-Programme wie der Texteditor SubEthaEdit [The10] zeigen ihre Vorteile auch nur bei funktionierender Netzwerkverbindung. Mit SubEthaEdit können sich Benutzer über das Bonjour-Protokoll im lokalen Netz oder über das Internet finden und gegenseitig dazu einladen, gemeinsam ein Dokument zu bearbeiten.

Die in den beiden vorhergehenden Absätzen vorgestellten Ansätze haben gemeinsam, dass sie entweder nur mit einer Internetverbindung funktionieren, oder die Konfliktbehandlung nur unterstützt wird, wenn von allen Clients gleichzeitig eine Verbindung zu einem Server hergestellt wird. Ansonsten muss das Zusammenführen der konflikthaften Versionen auch hier wieder manuell geschehen.

Bei manchen hier vorgestellten Programmen können die Tastaturanschläge der anderen Autoren live mitverfolgt werden. Dies wird nicht nur positiv bewertet. In [Man09] wird das Arbeiten mit Google Wave als „like talking to an overcurious mind reader“ beschrieben - das Bewusstsein, dass andere Benutzer einem „direkt beim Denken zusehen“, lähmt hierbei den eigenen Gedanken- und Arbeitsfluss. Auch bei Anwendungsfällen, bei denen Benutzer länger an einem einzelnen Dokument arbeiten und dadurch eine Art „Besitzanspruch“ entsteht, fällt es unter Umständen schwer, die Änderungen am eigenen Text live mitansehen zu müssen [Edl95].

Da die zu entwickelnde Anwendung durchaus auch für das Arbeiten an längeren Texten vorgesehen ist, kann auf das Feature „Live-Typing“ verzichtet werden.

3.4.3 Versionsverwaltungssysteme

Im Bereich der Softwareentwicklung ist schon lange der Einsatz von Versionsverwaltungssystemen wie Subversion [Apa10b] oder Git [Cha10] weit verbreitet. Mit solchen Systemen werden Änderungen an Dokumenten mitsamt Autor und Zeitstempel erfasst und in einzelnen *Commits* gespeichert. Die Versionen können später wiederhergestellt werden. Ebenfalls können mehrere Änderungen von unterschiedlichen Benutzern an einer einzigen Datei vom System automatisch zusammengeführt werden.

Ein solches System ist für reine Textdateien sehr zweckmäßig. Deshalb werden solche Programme hauptsächlich für Software-Verwaltung eingesetzt. Gängige Implementierungen haben aber kein Interface, das sich auch an weniger technisch versierte Menschen richtet. Die Umsetzung einer Anwendung mit beispielsweise einem Git-Backend ist aber eine erwägenswerte Option.

3.4.4 Datenbanken

Es liegt nahe, einen Ansatz mit Datenbanken, insbesondere den in Kapitel 2 vorgestellten neuen schemalosen Datenspeicher in Betracht zu ziehen. Einige Datenbanken oder Key-Value-Stores unterstützen Master-Master-Replikation und speichern die Daten auf der Festplatte. Diese kommen grundsätzlich für die Lösung der Aufgabe in Betracht.

Die dokumentenorientierte Datenbank CouchDB unterscheidet sich von den Alternativen dadurch, dass sie einen eigenen Webserver mitbringt. Mit diesem können nicht nur die Daten ausgeliefert, sondern auch in der Datenbank gespeicherte JavaScript-Dateien direkt ausgeführt werden. Dadurch kann die gesamte Anwendung in der Datenbank laufen. Das resultierende Programm ist dadurch automatisch von jedem Rechner bedienbar, auf dem CouchDB installiert ist, und der über einen Browser verfügt. Diese Eigenschaften bringt keine der anderen untersuchten Datenbanken mit.

Die freien relationalen Datenbanken PostgreSQL [Groa] und MySQL [Cora] können für Master-Master-Replikation zwischen zwei Mastern konfiguriert werden. Für PostgreSQL existieren eine Vielzahl an Erweiterungen [Grob], mit denen sich unter anderem ein Master-Master-Setup einrichten lässt. MySQL bedient sich dazu der Technik MySQL-Cluster [Cord], die Master-Master-Replikation mit einer Shared-Nothing-Architektur ermöglicht [Sto86, Kap. 1]. Unter [Max06] ist beschrieben, wie eine Replikation auch zwischen mehreren Knoten umgesetzt werden kann.

Der Key-Value-Store Riak [Bas10] hat ebenfalls ein HTTP-Interface und speichert seine Daten verteilt - es handelt sich dabei aber nicht um Peer-to-Peer-Replikation wie in CouchDB, sondern um ein Autobalancing für bessere Verfügbarkeit und *Performance (Leistung)* in größeren Systemen. MongoDB [10g10] unterstützt beschränkt Master-Master-Replikation und ermöglicht *Eventual Consistency* (vgl. Abschnitt 4.1.2), was sich für ein verteiltes Konzept anbietet. Die Fähigkeit von CouchDB, Anwendungslogik direkt in der Datenbank bzw. im Browser auszuführen, ist aber auch hier nicht vorhanden. Beide Technologien sind deshalb weniger gut als CouchDB für die Umsetzung der geplanten Anwendung geeignet.

Replikation und Clustering in den beschriebenen Systemen sind in etwa vergleichbar mit der Funktionalität von CouchDB-Lounge, die in Abschnitt 8.10 beschrieben ist. Automatische Markierung von Konflikten unterstützt ebenfalls keines der Systeme. Für die Umsetzung der Replikation innerhalb der Anwendungslogik bietet sich deshalb keiner dieser Ansätze an.

Im direkten Vergleich wird deutlich, dass sich CouchDB am besten für die Lösung der oben genannten Probleme eignet, da es Möglichkeiten zur Master-Master-Replikation, Konfliktbehandlung sowie ein passendes Konsistenz-Modell mitbringt, und Anwendungen direkt von der Datenbank ausgeliefert werden können. Im nächsten Abschnitt wird der gewählte Lösungsansatz noch näher beschrieben.

3.5 Beschreibung des gewählten Lösungsansatzes

Die Anwendung wird als lokales, aber netzwerkfähiges Programm erstellt. Die Daten werden dabei in einer CouchDB-Datenbank gespeichert, die Anwendungslogik wird als clientseitiges JavaScript im Browser ausgeführt. Der Funktionsumfang des Gliederungseditors wird mindestens das Erstellen und Löschen von Outlines umfassen; zeilenbasiert kann Text eingetragen und editiert werden; Zeilen werden beim Verlassen automatisch gespeichert und können ein- und ausgerückt werden.

Der Austausch der Daten sowie der Anwendung geschieht über die in CouchDB eingebaute Master-Master-Replikation. Hierbei dürfen die Daten auf allen Rechnern, die eine Kopie haben, gleichberechtigt verändert werden. Auf einem Server läuft eine weitere CouchDB-Instanz. Die Replikation zu diesem Server erfolgt automatisch, sobald von einem Benutzer dem Dokument etwas hinzugefügt wird. Weitere Benutzer können die gesamte Anwendung in die CouchDB-Installation auf ihrem Rechner herunterladen. Wenn eine Internetverbindung besteht, werden Updates an den Daten automatisch zum Server repliziert, und von ihm an weitere Benutzer weitergegeben, die gerade online sind. Die Anwendung benachrichtigt den Benutzer, sobald Änderungen vorliegen. Er kann sich diese dann durch ein Neuladen der Seite anzeigen lassen.

Die zentrale Aufgabe wird der Umgang mit Bearbeitungskonflikten in den Daten sein, die durch

das gleichzeitige Bearbeiten entstehen können. Gerade wenn ein Benutzer längere Zeit offline ist und dann repliziert, müssen durch Andere veränderte oder neu dazugekommene Zeilen eingefügt oder aktualisiert werden. CouchDB kann von sich aus auf aufgetretene Konflikte hinweisen. Die Entscheidung, wie diese Konflikte angezeigt und/oder automatisch gelöst werden können, muss aber beim Design der Anwendung getroffen werden. Da der begrenzte Bearbeitungszeitraum dies nicht zulässt, werden nicht alle möglicherweise auftretenden Konflikte berücksichtigt. Stattdessen werden einige Konfliktarten exemplarisch untersucht.

Des Weiteren werden Deployment und Skalierungsmöglichkeiten mit dem Clustering Framework CouchDB-Lounge und Amazon Elastic Compute Cloud (Amazon EC2) untersucht. Die Anwendung wird prototypisch deployt, Möglichkeiten zur Umsetzung einer hohen Verfügbarkeit des Servers werden beschrieben.

4 CouchDB - eine Datenbank für Verteilte Systeme

Nachdem der gewählte Lösungsansatz im vorherigen Kapitel begründet und kurz skizziert wurde, sollen in diesem und im nächsten Teil die für die Umsetzung der Aufgabenstellung verwendeten Technologien und Konzepte vorgestellt werden.

Im Mittelpunkt des Kapitels steht die ausführliche Darstellung der verwendeten Datenbank CouchDB. *Apache CouchDB („Cluster of unreliable commodity hardware Data Base")* ist ein dokumentenorientiertes Datenbanksystem [Apa09c]. CouchDB wird seit 2005 als Open-Source Projekt entwickelt und ist seit November 2008 ein offizielles Projekt der Apache Software Foundation [Apa09d]. CouchDB wurde ursprünglich in C++ geschrieben, wird aber seit 2005 größtenteils in der Programmiersprache *Erlang* [Eri10] entwickelt. Erlang wurde Ende der 80er Jahre des letzten Jahrhunderts für Echtzeitsysteme wie Telefonanlagen entworfen und zeichnet sich infolgedessen durch hohe Fehlertoleranz, Parallelität und Stabilität aus [Len09]. Das *Erlang/OTP*-System (*THe Open Telecom Platform*) umfasst neben der Programmiersprache Erlang auch Bibliotheken und das Laufzeitsystem.

Im ersten Teil dieses Kapitels werden die theoretischen Grundprinzipien von CouchDB erläutert. Der zweite Teil ist der Beschreibung der Datenbank aus der Sicht der Anwendungsentwicklerin gewidmet.

4.1 Theoretische Einordnung

Um die Motivation für die Entwicklung von CouchDB zu verstehen, wird zunächst kurz auf die jüngere Geschichte der Datenbanksysteme eingegangen und CouchDB im Hinblick auf die Drei-Schema-Architektur (s. Abschnitt 4.1.1) eingeordnet. Danach werden das CAP-Theorem und der Umgang von CouchDB mit nebenläufigen Transaktionen vorgestellt sowie die „RESTfulness" der HTTP-Schnittstelle untersucht. Des Weiteren wird der Unterschied im Ansatz von CouchDB im Vergleich zu traditionellen relationalen Datenbanken dargestellt. Dabei werden die einzelnen Aspekte von Aufbau, Eigenschaften und Funktionen angerissen, die dann im weiteren Verlauf dieses Kapitels ausführlich beschrieben werden.

4.1.1 Einordnung der Datenbankarchitektur

1975 entwarf das *Standards Planning and Requirements Committee (SPARC)* des *American National Standards Institute (ANSI)* ein Modell, das Anforderungen an den Aufbau eines Datenbanksystems beschreibt [Häo1]). Dieses Modell wird *ANSI-SPARC-Architektur* oder auch *Drei-Schema-Architektur* genannt.

Für die Benutzer einer Datenbank sollten Änderungen in den unteren Ebenen, also von Hardware, interner Speicherstruktur oder logischer Struktur, keine Auswirkungen haben [Cod83, S. 377]. Ist eine Datenbank nach der Drei-Schema-Architektur aufgebaut, wird die Sicht der Benutzer auf die Datenbank von der technischen Umsetzung getrennt. Die interne Datenspeicherung wird also transparent für die Benutzer.

Nach [Mato7, Kap. 1.2], lassen sich die drei Ebenen des Schemas wie folgt unterteilen:

Externe Schemata/Benutzersichten: Teilbereiche der Datenbank sind für verschiedene Benutzergruppen freigegeben. Hier können abgeleitete Daten eingetragen werden, ohne dass die zugehörigen Grunddaten sichtbar gemacht werden müssen.

Konzeptionelles Schema: Diese Ebene enthält die Beschreibung aller Datenstrukturen für die Datenbank, also die Datentypen und -verknüpfungen. Dabei ist unerheblich, auf welche Weise die Daten abgelegt werden. Das konzeptionelle Schema ist sehr stark von Datenbankentwurf und benutztem Datenmodell abhängig.

Internes Schema: Hier sind die Einzelheiten der physischen Datenspeicherung festgelegt, also die Aufteilung der Datenbank auf verschiedene Rechner oder Festplatten, oder Indizes zur Beschleunigung der Zugriffe.

Diese Architektur kann unabhängig von der Frage angewendet werden, ob das dem Datenbanksystem zugrunde liegende Datenbankmodell relational, objektorientiert, netzwerkartig oder an einem anderen Modell orientiert ist.

CouchDB lässt sich ebenfalls dem ANSI-SPARC-Standard gemäß beschreiben. Den externen Schemata entsprechen dabei die CouchDB-Views. Das konzeptionelle Schema ist hier die Repräsentation der Dokumente als JSON-Objekte, also die Gesamtansicht der Datenbank. Das interne Schema ist die Art und Weise der Datenspeicherung, die bei CouchDB über einen *B+-Baum (B+-Tree)* umgesetzt wird. Diese drei Schichten werden in späteren Abschnitten dieses Kapitels detailliert beschrieben.

4.1.2 Das CAP-Theorem

CAP steht für *Consistency (Konsistenz)*, *Availability (Verfügbarkeit)* und *Partition Tolerance (Partitionstoleranz)*. Bei der Modellierung von Verteilten Systemen ist der Begriff der *Partitionstoleranz* von großer Bedeutung [Var09, S. 62]. Er besagt, dass Subsysteme auch bei physikalischer Trennung und Verlust einzelner Nachrichten untereinander autonom weiter funktionieren können müssen. Eine Operation auf dem System muss auch dann erfolgreich durchgeführt werden, wenn ein Teil der Komponenten nicht erreichbar ist. Ein Verteiltes System muss jedoch noch zwei weitere Anforderungen erfüllen: *Konsistenz* ist gegeben, wenn alle Komponenten zur selben Zeit die gleichen Daten sehen. *Verfügbarkeit* bedeutet, dass das System auf jede Anfrage eine Antwort sendet, mit einer definierten und niedrigen Latenz. Die folgende Darstellung, sofern nicht anders angegeben, basiert auf [Gil02, S. 1-4] und [And10, Kap. 2].

Professor Brewer von der University of California hat mit dem *CAP-Theorem* die Annahme formuliert, dass die drei Eigenschaften Konsistenz, Verfügbarkeit und Partitionstoleranz zwar von Web Services erwartet werden, es aber in der Realität nur möglich ist, zwei der drei Ansprüche zu erfüllen [Bre00]. Da Partitionstoleranz bei Verteilten Systemen unabdingbar ist, muss beim Entwurf die Entscheidung zwischen Konsistenz und Verfügbarkeit getroffen werden.

In traditionellen Relationalen Datenbanksystemen (*RDBMS*) kann Konsistenz meist vorausgesetzt werden, da diese standardmäßig die *ACID*-Kriterien (*Atomizität, Konsistenz, Isolation, Dauerhaftigkeit*) erfüllen. Vollständige Konsistenz meint in diesem Kontext die Eigenschaft, dass auf einen Schreibzugriff folgende Lesezugriffe sofort auf die aktuellen Daten zugreifen können. Dies wird als *One-Copy-Serializability* oder auch *Strong Consistency* bezeichnet [Mos09]. In RDBMS wird dies durch *Locking*-Mechanismen erzwungen (s. Abschnitt 4.1.3). In einem Verteilten System, in dem Daten auf mehr als einem Rechner verteilt sind, gestaltet sich die Umsetzung von Konsistenz schwieriger. Verschiedene in den letzten Jahren entwickelte nichtrelationale Datenspeicher wie etwa Bigtable [Cha06], Hypertable [Hyp09], HBase [Apa10i], MongoDB [10g10] und MemcacheDB [Mem09] entscheiden sich trotzdem für die absolute Konsistenz und gegen eine Optimierung der Verfügbarkeit.

Andere Projekte wie Cassandra [Apa09a], Dynamo [Vog07], Project Voldemort [Pro] und CouchDB legen ihre Schwerpunkte stattdessen auf Verfügbarkeit. Dabei greifen sie auf unterschiedliche Strategien zurück, wie Konsistenz trotzdem umgesetzt werden kann. Durch einen sog. *Consensus Algorithm* wie *Paxos* [Lam01] kann garantiert werden, dass Komponenten auch dann zur gleichen Lösung für einen Konflikt kommen, wenn keine Verbindung zwischen ihnen besteht [Pea80]. Ein anderer Ansatz ist der Einsatz von *Time-Clocks*, mit denen eine Sortierung von Daten in Verteilten Systemen umgesetzt werden kann [Lam78].

Die Strategie von CouchDB unterscheidet sich von den meisten anderen, weil sie neben Verfügbarkeit und Partitionstoleranz *Eventual Consistency* vorsieht. Diese besagt, dass in einem

beschränkten Zeitfenster zwischen Schreib- und Lesezugriff auf ein Datum *Inkonsistenzen* (Widersprüche) auftreten können. Innerhalb dieses Zeitraums werden womöglich noch die alten Daten ausgegeben, danach jedoch spiegeln alle Lesezugriffe das Resultat des Schreibzugriffs wieder. Bei auftretenden Fehlern oder hoher Latenz können Datensätze also zeitweise inkonsistent erscheinen, die Konsistenz der Daten ist nur schlussendlich gegeben:

> The storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value. [Vog09]

Eventual Consistency ist nicht für alle Bereiche ein praktikables Konzept. Wenn Benutzereingaben stark aufeinander aufbauen, also die Eingaben voneinander abhängen und die Benutzer zeitweise auf veralteten Daten arbeiten, können sich Fehler kumulativ fortpflanzen und die Konsistenz des Gesamtsystems ist kompromittiert (bspw. im Finanzsektor). Mit CouchDB können daher ebenfalls Systeme mit Strong Consistency umgesetzt werden. Bei vielen Anwendungen jedoch ist es von größerer Bedeutung, dass ein Update jederzeit erfolgreich durchgeführt werden kann, ohne dass die Datenbank blockiert ist (bspw. bei einem *Social Network* oder beim vorliegenden Anwendungsfall).

4.1.3 Transaktionen und Nebenläufigkeit

Jedes Datenbanksystem, das für mehrere Benutzer ausgelegt ist, muss sich mit Fragen der Nebenläufigkeit beschäftigen. Beantwortet werden muss die Frage was passiert, wenn zwei Benutzer gleichzeitig versuchen, denselben Wert zu verändern. „Gleichzeitig“ meint hier nicht den exakt selben Zeitpunkt. Eine Operation, die Lesen, Ändern und Zurückspeichern eines Datums umfasst und die eine gewisse Zeit dauert, kann ein Problem mit Nebenläufigkeit verursachen, wenn ein anderer Benutzer eine ebensolche Operation innerhalb dieser Zeitspanne beginnt und dabei den vom ersten Benutzer zwischenzeitlich geänderten Wert überschreibt. Die Aufgabe der Datenbank ist es, eine solche Operation serialisierbar zu machen: Die beiden beschriebenen Operationen sollen dasselbe Ergebnis haben, wie wenn sie nacheinander stattgefunden hätten [Var09, S. 57]. Auch wenn es hier um sehr kurze Intervalle geht und Konfliktfälle in der Praxis unwahrscheinlich scheinen, müssen diese von vornherein in Design und Architektur einbezogen werden [Vog09].

Bei dem von RDBMS hauptsächlich benutzten Locking belegt eine Operation die Ressourcen, die sie ändern möchte, mit einer Sperre. Andere Operationen müssen auf die Aufhebung dieser Sperre warten, dann haben sie exklusiven Zugriff auf die Daten. Locking ist für nichtverteilte Datenbanksysteme eine Herangehensweise mit guter Performance [Var09, S. 57]. Operationen müssen nicht warten, nur weil sie nebenläufig sind. Andererseits ist Locking von einigem Overhead begleitet und schwer umzusetzen, wenn die Teilnehmer der Transaktion verteilt sind. Es existieren Protokolle, die auch in Verteilten Systemen Sperren setzen und auflösen können [Ber81], diese sind allerdings langsam und für die umzusetzende Anwendung unpraktikabel.

CouchDB verwendet daher zur Kontrolle von Nebenläufigkeit eine Umsetzung von *Optimistic Concurrency*, die sogenannte *Multi-Version Concurrency Control (MVCC)*:

> MVCC takes snapshots of the contents of the database, and only these snapshots are visible to a transaction. Once the transaction is complete, the modifications that were done are applied to the newest copy of the relation and the snapshot is discarded. This means that in any given time multiple different versions of the same data exists. [Par10, Kap. 2.1]

MVCC bringt Vorteile bei Verfügbarkeit und Performance, dafür sehen Benutzer teilweise inkonsistente Daten. Es gibt mehrere Wege, diesen Mechanismus umzusetzen. Entweder können mit Zeitstempeln oder Vektoruhren die Modifikationszeiten von Transaktionen und dadurch die Gültigkeit eines Updates bestimmt werden. Das Update wird dann entweder zugelassen oder zurückgewiesen. In [Lam78] ist dies näher beschrieben. Bei CouchDB werden den Objekten keine Vektoren zugewiesen, sondern eine UUID und eine Revisionsnummer. Außerdem verwendet CouchDB das *Copy-On-Write*-Verfahren, bei dem zwei Prozesse nie auf einen Eintrag zugreifen, sondern ihn hintereinander in ein Log schreiben. Felix Hupfeld beschreibt in [Hup04, Kap. 2.2] einen *Log-based Storage Mechanismus*, siehe auch [Joh09]:

> The basic organization of a log structured storage system is, as the name implies, a log - that is, an append-only sequence of data entries. Whenever you have new data to write, instead of finding a location for it on disk, you simply append it to the end of the log.

Genau dieser Mechanismus wird in CouchDB angewendet, wenn die Versionen von Dokumenten in Revisionen oder auch die Daten im B+-Baum gespeichert werden. Dies wird in Abschnitt 4.2.7 genauer erklärt.

Der Nachteil von MVCC ist, dass zusätzliche Schichten von Komplexität in der Anwendungslogik bearbeitet werden müssen, die ein RDBMS still behandeln würde. CouchDB bietet hierfür Möglichkeiten, die in Abschnitt 4.2.1 erklärt werden. In [Var09, S. 60] wird dies jedoch als Vorteil diskutiert: Mit RDBMS werden Anwendungen so entworfen, dass bei steigendem Durchsatz leicht *Bottlenecks* (Engpässe) entstehen, die dann nicht mehr beseitigt werden können. MVCC unterstützt die Entwickler darin, von Anfang an mögliche Konfliktquellen sauber zu behandeln. Dadurch wird ein Zuwachs der Zugriffszahlen die Performance der Anwendung weniger wahrscheinlich beeinträchtigen. Auch unter hoher Last kann die Rechenleistung des Servers voll ausgelastet werden, ohne dass auf gesperrte Ressourcen gewartet werden muss, Requests werden parallel ausgeführt.

4.1.4 Replikation

Allgemein werden mithilfe von Replikation Daten zwischen Komponenten eines Verteilten Systems synchronisiert. Bei CouchDB bedeutet dies, dass der Inhalt einer Datenbank in eine andere übertragen wird; Dokumente, die in beiden Datenbanken existieren, werden auf denselben Stand gebracht. Replikation lässt sich anhand mehrerer Dimensionen einteilen:

4.1.4.1 *Conservative* oder *Optimistic Replication*

Die vielleicht wichtigste Designentscheidung, die bei dem Konzept Replikation getroffen werden muss, ist die Frage nach dem Umgang mit nebenläufigen Updates: Wie soll sich das System verhalten, wenn verschiedene Repliken dasselbe Datum gleichzeitig aktualisieren wollen? Auf diese Frage wurde bereits in Abschnitt 4.1.3 näher eingangen. [Guy98, Kap. 2] nennt zwei Herangehensweisen: Bei *Conservative* oder nach [Sai05] *Pessimistic Replication* muss die Konsistenz vor jedem Update überprüft werden. Ein Update wird abgelehnt, wenn es nebenläufig stattfindet. Für den vorliegenden Anwendungsfall ist diese Strategie nicht umsetzbar, da die Repliken nicht dauerhaft verbunden sind. Stattdessen setzt CouchDB eine *Optimistic Replication* um [Sai05, S. 43].

Bei Optimistic Replication werden Änderungen an replizierten Datensets akzeptiert, ohne dass die Repliken sich im gleichen Moment darüber abstimmen müssen oder ein Locking-Mechanismus eingesetzt wird. Die dadurch entstehenden Konflikte an den replizierten Daten zu bearbeiten ist Sache des Systems. CouchDB unterstützt dies durch automatische Konflikterkennung und -markierung, dies wird in Abschnitt 4.2.1 beschrieben.

4.1.4.2 *Client-Server-* oder *Peer-to-Peer-Modell*

Nach [Guy98, Kap. 2] wird bei Replikation nach dem *Client-Server-Modell* ein Update zuerst einem Server mitgeteilt, der es dann an alle Clients ausliefert. Das System wird dadurch simpler. Allerdings ist das System an einen nie ausfallenden Server gebunden. Replikation nach dem *Peer-to-Peer-Modell* erlaubt es den Repliken, sich gegenseitig ihre Updates mitzuteilen. Dadurch können Updates zum einen schneller verbreitet werden: Sobald Konnektivität vorhanden ist, egal zwischen welchen Komponenten, kann diese genutzt werden. Eine CouchDB-Installation kann mit jeder anderen CouchDB-Instanz in beide Richtungen Master-Master-Replikation betreiben und ist daher für beide Modelle geeignet. Welche Strategie umgesetzt werden soll, hängt vom konkreten Anwendungsfall ab.

4.1.4.3 Benachrichtigungstrategien

Bei den Benachrichtigungstrategien [Guy98, Kap. 2] wird zwischen *Immediate Propagation* und *Periodic Reconsiliation* unterschieden. Bei Immediate Propagation werden die anderen Repliken sofort nach einem Update benachrichtigt. Bei Periodic Reconsiliation werden die Repliken regelmäßig, zu einer passenden Zeit, über stattgefundene Updates benachrichtigt. CouchDB unterstützt beide Benachrichtigungstrategien. Da das hier zu erstellende System einen Betrieb auch ohne Netzverbindung erlaubt, muss es eine Form von Periodic Reconsiliation unterstützen, da Immediate Propagation fehlschlagen wird, wenn Knoten offline sind.

4.1.4.4 *Eager* oder *Lazy Replication*

Weiterhin nimmt Jim Gray in [Gra96, Kap. 1] eine Einteilung in *Eager Replication* und *Lazy Replication* vor. Bei ersterer werden alle Repliken immer sofort aktualisiert, sie müssen also immer verbunden sein. Dies ist für den in dieser Arbeit behandelten Anwendungsfall nicht praktikabel:

> In Systemen, die über Weitverkehrsnetze kommunizieren oder mobile Endgeräte einschließen, muß das Replikationssystem mit großen Kommunikationslatenzen umgehen können. Deshalb werden in solchen Systemen in der Regel nur asynchrone Replikationsalgorithmen [...] eingesetzt. [Hup09, S. IV]

Die Replikation von CouchDB ist demnach „lazy“ - Updates werden asynchron verbreitet. Durch den Replikationsalgorithmus von CouchDB kann eine CouchDB-Instanz die Änderungen einer anderen dann anfordern, wenn zwischen beiden eine Verbindung besteht.

4.1.5 HTTP-Schnittstelle

In [KT04] wird für dezentrale und unabhängige Verteilte Systeme der Architekturstil *REST (Representational State Transfer)* empfohlen. REST-konform oder *RESTful* ist eine Schnittstelle, mit der über HTTP [Fie96] Daten übertragen werden können, wenn jede Ressource mit einer eigenen URL angesprochen werden kann [Fie00]. Weitere Vorgaben sind die Zustandslosigkeit des Protokolls, wohldefinierte Operationen, und die Möglichkeit, unterschiedliche Repräsentationen einer Ressource anzufordern.

Nicht alle *APIs (Programmierschnittstellen)*, die RESTful genannt werden, die also angeblich dem REST-Architekturstil entsprechen, sind zurecht so eingeordnet. In der von NORD Software Consulting vorgenommenen Klassizifierung von HTTP-basierten APIs [NOR] wird zwischen verschiedenen Stufen von RESTfulness unterschieden. Die meisten APIs fallen demnach in die Kategorien „HTTP-based Type I“, „HTTP-based Type II“ oder „REST“. APIS, die in die ersten

beiden Kategorien eingeordnet sind, verletzen eine der REST-Einschränkungen, da Client und Server durch das Schnittstellendesign fest aneinander gekoppelt sind.

Dies trifft auch auf die API von CouchDB zu, obwohl diese z. B. in [Apa09b] als RESTful bezeichnet wird. Nach [NOR10] muss eine RESTful API keine differenzierte Dokumentation enthalten, stattdessen würde eine Aufzählung der verfügbaren Medientypen und Felder genügen. Die CouchDB-API kann nach dieser Studie auch nicht in die Kategorie HTTP-based Type II eingeordnet werden, da ein generischer Medientyp verwendet wird, der die Ressourcen nicht selbsterklärend macht. Da die API allerdings korrekt bezeichnete Methodennamen in den URIs verwendet, kann sie als HTTP-based Type I bezeichnet werden.

In [And10, Kap. 4] wird die eingeschränkte RESTfulness von manchen Teilen der API bestätigt. Beispielsweise ähnelt die API für die Replikation traditionellen *Remote Procedure Calls*. Eine ausschließlich lose Kopplung, wie es die REST-Architektur vorsieht, ist bei einer Datenbank-API jedoch nicht unbedingt nötig [NOR10]. Trotzdem kann die API von CouchDB mithilfe der in Abschnitt 4.2.5 erwähnten *show*- und *list*-Funktionen auch HTTP-based Type II- und REST-konform gemacht machen.

4.1.6 Abgrenzung zu relationalen Datenbanksystemen

Das relationale Datenmodell wurde Anfang der 70er Jahre von Edgar Codd [Cod83] erstmals wissenschaftlich beschrieben. IBM und Oracle implementierten Ende der 70er Jahre die ersten darauf basierenden Datenbanksysteme. Datenbanken liefen zu dieser Zeit noch auf einzelnen, nicht vernetzten Großrechnern. Diese mussten regelmäßig größere Operationen ausführen, die viel Datenbanklogik erforderten [Leh08]. Bei jeder dieser Operationen datenbankweit die Konsistenz zu überprüfen stellte kein Problem dar, da die Operationen einfach nacheinander abgearbeitet wurden [And10, Kap. 2]. Daten wurden durch physische Backups gegen Verlust abgesichert, Replikation kam erst später auf. RDBMS sind für eine solche Nutzungsweise optimiert.

4.1.6.1 Replikation und Konfliktbehandlung

Ab Anfang der 80er Jahre setzten sich relationale Datenbanksysteme auch in anderen Anwendungsbereichen immer mehr durch. Die Einsatzszenarien sehen allerdings heute oft anders aus als damals: Im Bereich der Internetanwendungen müssen Server meist eine Vielzahl einzelner Abfragen gleichzeitig abarbeiten. Das Verhältnis zwischen Komplexität der Abfragen und Anzahl der Zugriffe hat sich stark verändert. Hinzu kommt die bei Verteilten Systemen unweigerlich aufkommende Frage nach der Umsetzung von Replikation und Konfliktlösungsstrategien.

Trotz dieser Nachteile wird zur Implementierung von Webanwendungen heute die relationale Datenbank MySQL [Cora] mit Abstand am häufigsten eingesetzt [Alf08, S. 18]. Replikation bei

MySQL ist nach dem Prinzip des *Log Replay* aufgebaut [Corc]. In einem Master-Master-Setup treten jedoch regelmäßig nebenläufige, sich widersprechende Schreibzugriffe auf. Wenn die Datenbank Inkonsistenzen nicht definiert behandelt, müssen die Konflikte mit selbst zu erstellenden Datenbankfunktionen oder Anwendungslogik gelöst werden [Max06]. Die Replikationsstrategie von CouchDB dagegen ist inkrementell, auch bei Verbindungsabbruch während des Replikationsvorgangs bleiben die Daten stets in einem konstanten Zustand. Dies wird in Abschnitt 4.2.3 erläutert. Die Fähigkeit von CouchDB, mit nebenläufigen Updates und entstandenen Konflikten umzugehen, wird in Abschnitt 4.2.1 dargelegt, und ist in Abschnitt 4.2.7 durch die Darstellung der Implementierung von CouchDB erklärt.

4.1.6.2 Keine Middleware

CouchDB wurde entwickelt, um den veränderten Ansprüchen und heutigen Anforderungen an eine Datenbank für Webanwendungen gerecht zu werden - „[it is] built *of* the web“ [Kap07]. Die mit CouchDB umgesetzten Konzepte sind nicht neu. In [Ass98, S. 39] wurde beschrieben, wie sich statische Webanwendungen der ersten Generation zum bisher verbreiteten Modell weiterentwickelten: Datenbank- und Anwendungsentwicklung wird völlig getrennt vorgenommen, zur Kommunikation mit den Anwendern werden Interfaces wie *CGI* [W3C09] eingesetzt. Der Einsatz von Middleware ist dabei nötig, um ...

> ... ausgehend von den vorhandenen Schnittstellen gängiger Web-Server und Datenbanksysteme den Übergang zwischen den Systemen für einen Entwickler so einfach und transparent wie möglich [zu] gestalten. [Ass98, S. 24]

Für die Zukunft der sog. Internetdatenbanken wurde in [Ass98, S. 39] vorausgesagt, dass diese nicht nur Zugriff auf die Daten ermöglichen, sondern auch die Interaktion mit dem Anwendungssystem einbeziehen müssen. Bei einer mit CouchDB erstellten Anwendung kann diese Middleware entfallen. Stattdessen kann ein Anwendungsprogramm direkt mit der Datenbank kommunizieren. Dies geschieht über eine HTTP-API, die in Abschnitt 4.2.2 vorgestellt wird. Auf diese Weise können stabile Anwendungen mit vergleichsweise geringem Aufwand umgesetzt werden.

4.1.6.3 Schemalosigkeit

Viele der Probleme beim Entwurf einer modernen Webanwendung (vgl. Kapitel 3 und 1.1) beinhalten unvorhersehbares Verhalten der Benutzer und Input von einer großen Menge von Menschen mit einer großen Menge von Daten [Var09, S. 36]. Dies umfasst beispielsweise die Suche im Internet, das Erstellen von Graphen in Social Networks, Auswertung von Kaufgewohnheiten etc. Diese Aufgaben bringen oft unübersichtliche Datenstrukturen mit sich, die vorher nur schwer

genau definiert und modelliert werden können. Laut [Bar09] sind solche Daten für die Abbildung als relationale Datenstrukturen nicht gut geeignet:

> RDBMSs are designed to model very highly and statically structured data which has been modeled with mathematical precision. Data and designs that do not meet these criteria, such as data designed for direct human consumption, lose the advantages of the relational model, and result in poorer maintainability than with less stringent models.

Eine dokumentenorienterte Datenbank besteht aus einer Reihe von unabhängigen Dokumenten; alle Daten für ein Dokument sind in diesem selbst enthalten:

> In fact, there are no tables, rows, columns or relationships in a document-oriented database at all. This means that they are schema-free; no strict schema needs to be defined in advance of actually using the database. If a document needs to add a new field, it can simply include that field, without adversely affecting other documents in the database. [Len09]

Die Schemalosigkeit von CouchDB ist für die Umsetzung des Prototypen zwar relevant, für die Konzeption aber nicht zentral. Dies kann sich aber in späteren Versionen der Anwendung anders darstellen, wenn der Gliederungseditor mehrere Spalten mit unterschiedlichen Datentypen und Medien enthalten wird (vgl. Kapitel 10).

4.1.6.4 Unique Identifiers

In relationalen Datenbanken werden Zeilen einer Tabelle üblicherweise mit einem Primärschlüssel identifiziert, der oft durch eine *auto-increment*-Funktion bestimmt wird [Len09]. Eindeutig sind diese Schlüssel jedoch nur für die Datenbank und die Tabelle, in der sie erzeugt wurden. Wenn auf zwei unterschiedlichen Datenbanken, die später synchronisiert werden, ein Eintrag hinzugefügt wird, wird hier ein Konflikt auftreten [Corb]. In CouchDB wird jedem Dokument bei der Erstellung eine *UUID (Universally Unique Identifier)* zugewiesen. Auf diese Weise wird ein Konflikt statistisch nahezu unmöglich. Ein Überblick über Dokumente in CouchDB findet sich in Abschnitt 4.2.1.

4.1.6.5 Views statt Joins

Einer der wichtigsten Unterschiede zwischen dokumentenorienterten und relationalen Datenbanken ist die Art und Weise, wie Abfragen an die Datenbank gestellt werden. Da CouchDB keine Primär- und Fremdschlüssel kennt, können Daten nicht direkt verknüpft und über Joins abgerufen werden. Stattdessen kann mithilfe von *Views* eine Beziehung zwischen beliebigen

Dokumenten der Datenbank hergestellt werden, ohne dass diese Beziehung in der Datenbank vordefiniert sein muss. Views werden in Abschnitt 4.2.6 erklärt. Oft werden Joins auch schon durch die Modellierung der Daten in Dokumenten überflüssig gemacht.

4.2 Beschreibung

In diesem Abschnitt werden die Features und einige Implementierungsdetails von CouchDB vorgestellt. In einem CouchDB-Datenbanksystem können beliebig viele Datenbanken angelegt werden, in denen Dokumente enthalten sind. Die Administrationsoberfläche *Futon* kann in einem Browser unter der URL `http://localhost:5984/_utils` besucht werden. In Abbildung A.4 findet sich ein Screenshot von einer CouchDB-Instanz und den darin enthaltenen Datenbanken, Abbildung A.5 zeigt den Inhalt einer Datenbank. Operationen auf der Datenbank werden entweder über diese Oberfläche oder programmatisch vorgenommen.

Mit CouchDB lässt sich eine differenzierte Zugriffskontrolle mit Benutzerverwaltung und Rechtevergabe umsetzen. Dies wird aus Platzgründen in dieser Arbeit nicht behandelt. Ebenfalls werden in dieser Darstellung manche Datenbank-Funktionen sowie einige Teile der HTTP-API ausgespart. Die Informationen in den folgenden Abschnitten, soweit nicht anders angegeben, können in [And10], [Apa09b] und [Len09] nachgelesen werden.

4.2.1 Dokumente und Konfliktbehandlung

In CouchDB-Dokumenten werden die eigentlichen Datensätze als JSON-Objekte (siehe auch Abschnitt 5.1.3.1) gespeichert. Ein Dokument kann eine beliebige Anzahl von beliebig großen Feldern haben, die einen innerhalb des Dokuments eindeutigen Namen tragen müssen. Binäre Attachments können ebenfalls an ein Dokument angehängt werden. Ein Dokument ist mit einer eindeutigen ID (`_id`) versehen, die beim Erstellen angegeben oder als UUID zu mathematisch nahezu 100 Prozent eindeutig erzeugt wird. Abbildung A.6 zeigt beispielhaft ein Dokument.

Als weiteres Metadatum enthält das Dokument eine Revisionsnummer, genannt `_rev`. Werden Änderungen an einem Dokument vorgenommen, wird dieses nicht verändert; stattdessen wird eine neue Version des gesamten Dokuments erzeugt, das bis auf die vorgenommenen Änderungen und die neue Revisionsnummer identisch ist. Auf diese Weise beinhaltet die Datenbank eine komplette Versionsgeschichte jedes Dokuments. Mit dieser Art der Datenspeicherung implementiert CouchDB ein *lockless and optimistic document update model* (s. Abschnitt 4.1.3).

Ein Dokument wird nach dem folgenden Muster geändert: Das Dokument wird vom Client geladen, verändert und mit Angabe von ID und Revision in der Datenbank gespeichert. Wenn ein anderer Client inzwischen seine Änderungen am selben Dokument gespeichert hat, wird der erste

Client beim Speichern eine Fehlermeldung bekommen (HTTP Status-Code 409: „conflict“). Um diesen aufzulösen, muss die aktuelle Version des Dokuments noch einmal geladen und modifiziert werden, bevor ein neuer Speicherversuch gemacht werden kann. Dieser schlägt entweder komplett fehl oder wird vollständig durchgeführt, zu keinem Zeitpunkt werden unvollständige Dokumente gespeichert.

Das Löschen eines Dokuments verläuft auf eine ähnliche Weise. Vor dem Löschen muss die aktuellste Version des Dokuments vorliegen. Der eigentliche Löschvorgang besteht darin, dem Dokument das Feld `_deleted=true` hinzuzufügen. Gelöschte Dokumente werden genau wie ältere Versionen aufbewahrt, bis die Datenbank kompaktiert wird.

Konflikte können dennoch auftreten, wenn an einem replizierten Datenset unabhängig voneinander Updates vorgenommen wurden. CouchDB verfügt allerdings über eine automatische Konflikterkennung, wodurch die Konfliktbehandlung vereinfacht wird. Die in beiden Kopien geänderten Dokumente werden ähnlich wie in einem Versionskontrollsystem beim Zusammenführen automatisch als konflikthaft gekennzeichnet. Dafür wird ihnen ein Array namens `_conflicts` hinzugefügt, in dem alle konflikthaften Revisionen gespeichert sind. Durch einen deterministischen Algorithmus wird eine der Revisionen als die „Gewinnerversion“ gespeichert. Nur diese wird in den Views angezeigt. Die andere Revision wird in der Geschichte des Dokuments gespeichert, auf sie kann noch zugegriffen werden. Es ist Aufgabe der Anwendung bzw. der Benutzer, die Konflikte aufzulösen; dies kann durch Zusammenführen, Rückgängig machen oder Akzeptieren der Änderungen geschehen. Auf welcher Replik dies geschieht, ist unerheblich, solange am Ende der gelöste Konflikt durch Replikation allen Kopien bekannt gemacht wird.

4.2.2 HTTP-Schnittstelle

Die Daten aus einer CouchDB-Datenbank können über eine API abgefragt und geschrieben werden. Diese API ist über die HTTP-Methoden GET, POST und PUT ansprechbar. Die Daten werden als JSON-Objekte zurückgegeben. Da JSON und HTTP von vielen Sprachen und Bibliotheken unterstützt werden, können von einer beliebig umgesetzten Anwendung aus Datenbankoperationen auf CouchDB vorgenommen werden.

Für die JavaScript-HTTP-API von CouchDB wurde im Verlauf dieser Arbeit eine Testsuite erstellt, dies wird in Abschnitt 8.8.3 erläutert.

Die CouchDB-API verfügt über eine Reihe von Funktionen, die sich nach [And10, Kap. 4] in vier Bereiche unterteilen lassen. Für jeden der Bereiche werden einige Einsatzmöglichkeiten und ein Beispiel für die Abfrage mit dem Kommandozeilen-Werkzeug *cURL* genannt.

4.2.2.1 Server-API

Wird ein einfacher GET-Request an die URI des CouchDB-Servers gerichtet, sendet dieser die Versionsnummer der CouchDB-Installation: `curl http://localhost:5984/` liefert das JSON-Objekt `{"couchdb":"Welcome","version":"0.11.0b902479"}` zurück. Mit anderen Funktionen können eine Liste aller Datenbanken abgefragt oder Konfigurationseinstellungen vorgenommen werden. Benutzeridentifikation wird ebenfalls direkt über den Server abgewickelt, Benutzer können sich gegenüber dem Server identifizieren oder ausloggen.

4.2.2.2 Datenbanken-API

Mit dem Befehl `curl -X PUT http://localhost:5984/exampledb` kann eine Datenbank erstellt werden. Im Erfolgsfall wird hier `{"ok":true}` zurückgegeben. Schlägt das Anlegen fehl, weil eine Datenbank mit diesem Namen bereits existiert, erhält man eine Fehlermeldung. Datenbanken können ebenfalls gelöscht, kompaktiert oder es können Informationen über sie abgefragt werden. Ein neues Dokument kann durch `curl -X POST http://localhost:5984/exampledb -d '{"foo":"bar"}'` angelegt werden. CouchDB gibt als Antwort die ID und die Revision des angelegten Dokuments zurück: `{"ok":true,"id":"6651b95e15b411dbab3d2a7a7d000452","rev":"1-303d5e305201766b21a42747173681d6"}`. Wird statt POST die Methode PUT verwendet, kann die ID des zu erstellenden Dokuments selbst gewählt werden.

4.2.2.3 Dokumenten-API

Mit einem GET-Request auf die URI des Dokuments (`http://localhost:5984/exampledb/6651b95e15b411dbab3d2a7a7d000452`) kann das Dokument aus dem obigen Beispiel wieder angefordert werden. Das Dokument kann durch einen PUT-Request geändert werden. Dabei muss die ID, das komplette geänderte Dokument und die letzte Revision des Dokuments mit angegeben werden. Fehlt die Revision oder ist sie unkorrekt, schlägt das Update fehl.

4.2.2.4 Replikations-API

Mit dem Befehl `curl -X POST http://127.0.0.1:5984/_replicate -d '{"source":"exampledb", "target":"exampledb-replica"}'` wird Replikation zwischen zwei Datenbanken gestartet. Soll eine Replikation in beide Richtungen realisiert werden, wird der Befehl ein zweites mal mit vertauschter Quelle und Ziel aufgerufen. Auf Parameter wird im nächsten Abschnitt näher eingegangen.

4.2.3 Replikation

Damit eine Replik sofort von Updates auf anderen Repliken erfährt, kann die Replikation mit der Option `continuous=true` gestartet werden. Dieser Mechanismus wird *Continuous Replication* genannt. Dadurch wird die HTTP-Verbindung offen gehalten, und jede Änderung an einem Dokument wird automatisch sofort repliziert. Continuous Replication muss explizit neu gestartet werden, wenn eine Netzwerkverbindung wieder verfügbar wird. Auch nach einem Server-Neustart wird sie nicht automatisch fortgesetzt.

Es werden nur die Daten repliziert, die seit dem letzten Replikationsvorgang geändert wurden. Der Prozess ist also inkrementell. Wenn die Replikation durch Netzwerkprobleme oder plötzliche Ausfälle von Knoten fehlschlägt, wird der nächste Replikationsvorgang an der Stelle fortgesetzt, an der der letzte unterbrochen wurde. Replikation kann durch sogenannte *Filter-Funktionen* gefiltert werden, so dass nur bestimmte Dokumente repliziert werden.

4.2.4 Change Notifications

CouchDB-Datenbanken haben eine Sequenznummer, die bei jedem Schreibzugriff an die Datenbank inkrementiert wird. Gespeichert werden auch die Änderungen zwischen zwei Sequenznummern. Dadurch können Unterschiede zwischen Datenbanken effizient festgestellt werden, wenn Replikation nach einer Pause wieder aufgenommen wird. Diese Unterschiede werden nicht nur intern für die Replikation benutzt, sie können in Form des *Changes-Feeds* auch für Anwendungslogik benutzt werden.

Mit dem Changes-Feed kann die Datenbank auf Änderungen überwacht werden. Die Abfrage von `http://localhost:5984/exampledb/_changes` gibt ein JSON-Objekt zurück, das sowohl die aktuelle Sequenznummer als auch eine Liste aller Änderungen der Datenbank enthält:

```
{"results": [
    {"seq":1,"id":"test","changes":[{"rev":"1-aaa8e2a031bca334f50b48b6682fb486"}]},
    {"seq":2,"id":"test2","changes":[{"rev":"1-e18422e6a82d0f2157d74b5dcf457997"}]}
  ],
"last_seq":2}
```

Der Changes-Feed kann mit unterschiedlichen Parametern versehen werden. So können mit `since=n` nur Änderungen seit einer bestimmten Sequenznummer angefordert werden. Mit `feed=continuous` kann der Feed, ähnlich wie die Replikation, so konfiguriert werden, dass er nach jeder Änderung in der Datenbank einen Eintrag zurückgibt. Die bereits erwähnten *Filter-Funktionen* ermöglichen, nur Dokumente mit bestimmten Änderungen zurückzugeben.

4.2.5 Anwendungen mit CouchDB

In Dokumenten kann auch Programmcode enthalten sein, der von CouchDB ausgeführt werden kann. Solche Dokumente werden *Designdokumente* genannt. Üblicherweise ist jeder Anwendung, die von CouchDB ausgeliefert werden soll, ein Designdokument zugeordnet.

Ein Designdokument ist nach festen Vorgaben strukturiert. Im Folgenden findet sich eine Auflistung der in einem Designdokument typischerweise enthaltenen Dokumente:

_id: Enthält das Präfix `_design/` und den Namen des Designdokuments bzw. der Anwendung, z. B. `_design/doingnotes`.

_rev: Von Replikation und Konfliktbehandlung werden Designdokumente behandelt wie normale Dokumente, deswegen enthalten sie eine Revisionsnummer.

_attachments: In diesem Feld enthaltener Programmcode kann clientseitig im Browser ausgeführt werden. Hier kann die Anwendungslogik für eine CouchDB-Applikation enthalten sein.

_views: Ebenso wie `list-`, `show-` und `filter`-Felder enthält dieses Feld Funktionen, mit denen der Inhalt einer Datenbank gefiltert, strukturiert und/oder modifiziert ausgegeben werden kann. Dieser Code wird serverseitig, also von der Datenbank ausgeführt.

Abbildung A.7 enthält einen Screenshot von einem in Futon geöffneten Designdokument.

4.2.6 Views

In relationalen Datenbanken werden die Beziehungen zwischen Daten ausgedrückt, in dem „gleiche“ Daten in einer Tabelle gespeichert werden, und zusammengehörige Daten mit Primär- und Fremdschlüsseln verknüpft sind. Aufgrund dieser Beziehungen können dann durch dynamische Abfragen aggregierte Datensets angefordert werden. CouchDB wählt hier einen gegenteiligen Ansatz. Auf Datenbankebene sind keine Verknüpfungen realisiert. Verbindungen zwischen Dokumenten können auch dann noch gezogen werden, wenn die Daten schon vorhanden sind. Dafür werden die Abfragen dieser Daten und ihre Ergebnisse statisch in der Datenbank gespeichert. Sie haben keine Auswirkungen auf die Dokumente in der Datenbank. Diese Abfragen werden als Indizes gespeichert, die *Views* genannt werden.

Views werden in Designdokumenten abgelegt und beinhalten JavaScript-Funktionen, die Abfragen mithilfe von *MapReduce* [Yano7] formulieren. Eine *Map-Funktion* bekommt nacheinander alle Dokumente als Argument übergeben und bestimmt anhand dessen, ob es oder einzelne Felder durch den View verfügbar gemacht werden sollen. Wenn eine View eine *Reduce-Funktion* enthält, wird diese zum Aggregieren der Ergebnisse verwendet. Listing 4.1 zeigt eine View, mit

der alle Dokumente auf das Feld `kind` mit dem Wert `Outline` überprüft werden. Wenn ein Dokument solch ein Feld enthält, wird dem resultierenden JSON-Objekt ein Eintrag hinzugefügt, der als Schlüssel die Dokumenten-ID und als Wert das Dokument enthält.

```
1 function(doc) {
2   if(doc.kind == 'Outline') {
3     emit(doc._id, doc);
4   }
5 }
```

Listing 4.1: View: Map-Funktion zum Ausgeben aller Outlines

Wird diese View unter dem Namen `outlines` in einem Designdokument `designname` gespeichert, kann sie mithilfe eines HTTP GET-Requests auf die URI `http://localhost:5984/exampledb/_design/designname/outlines` abgerufen werden. Views werden bei ihrem ersten Abrufen erstellt und dann mitsamt ihrem Index, wie auch normale Dokumente, in einem B+-Baum gespeichert. Werden weitere Dokumente hinzugefügt, die im Ergebnis der View enthalten sind, werden sie automatisch zur gespeicherten View hinzugefügt, wenn diese das nächste mal abgerufen wird.

Der Zugriff auf eine View kann über *Schlüssel (Keys)* und *Schlüsselbereiche (Key Ranges)* eingegrenzt werden. Die Abfrage von `http://localhost:5984/exampledb/design/designname/outlines?key="5"` gibt nur das Outline mit der ID `5` zurück. Mit `http://localhost:5984/exampledb/design/designname/outlines?startkey="2"&endkey="7"` können alle Outlines angefordert werden, deren ID zwischen `2` und `7` liegt. Es existieren eine Vielzahl weiterer Parameter, mit denen die Abfrage weiter präzisiert werden kann. Die Schlüssel bzw. Schlüsselbereiche werden direkt auf den Datenbankengine gemappt, dadurch sind die Zugriffe sehr performant. Dies wird im nächsten Abschnitt erklärt.

4.2.7 Implementierung

Eine CouchDB-Datenbank ist immer in einem konsistenten Zustand, auch wenn der CouchDB-Server mitten in einem Speichervorgang abstürzen sollte. Dies wird in diesem Abschnitt mit einer Charakterisierung des verwendeten Datenbankengines begründet. Die Darstellung stützt sich auf [Ho,08] und [And10, Kap. G].

Die eingesetzte Datenstruktur ist ein *B+-Baum*, eine Variation des *B-Baums*. Ein B+-Baum ist auf die Speicherung von großen Datenmengen und schneller Abfrage dieser ausgerichtet. Auch für *„extremely large datasets“* wird eine Zugriffszeit von unter 10 Millisekunden garantiert [Bay08]. B+-Bäume wachsen in die Breite; auch mit mehreren Millionen Einträgen haben sie gewöhnlich eine einstellige Tiefe. Dies ist vorteilhaft, da CouchDB als Speichermedium Festplatten verwendet, wo jeder Traversierungsschritt ein zeitintensiver Vorgang ist.

Ein B+-Baum ist ein vollständig balancierter Suchbaum, in dem Daten nach Schlüsseln sortiert gespeichert werden [Bayo8]. In einem Knoten können mehrere Schlüsselwerte enthalten sein. Jeder Knoten verweist auf mehrere Kindknoten. Bei CouchDB sind die eigentlichen Daten ausschließlich in den Blättern gespeichert. In den B+-Bäumen werden sowohl die Dokumente als auch die Views indiziert. Dabei wird für jede Datenbank und jede View ein eigener B+-Baum angelegt.

Pro Datenbank ist nur ein gleichzeitiger Schreibzugriff erlaubt, die Schreibzugriffe werden serialisiert. Lesezugriffe können nebenläufig zueinander und zu Schreibzugriffen stattfinden. Datenbanken und Views können demnach gleichzeitig abgefragt und erneuert werden. Da CouchDB seine Daten im *Append-Only-Modus* ablegt, enthält die Datenbankdatei eine komplette Versionsgeschichte aller Dokumente. MVCC kann dadurch effektiv umgesetzt werden.

Wie in Abschnitt 4.2.1 beschrieben, wird beim Ändern eines Dokuments dieses nicht überschrieben, sondern eine neue Revision erstellt. Danach werden die Knoten des B+-Baums nacheinander aktualisiert, bis sie alle auf den Speicherort der neusten Version des Dokuments verweisen. Dies geschieht ausgehend vom Blatt des Baums, der das Dokument enthält, bis hoch zum Wurzelknoten. Dieser wird also am Ende jedes Schreibzugriffs modifiziert. Wenn ein Lesezugriff noch die Referenz auf den alten Wurzelknoten hat, verweist dieser zu einer veralteten, aber konsistenten Momentaufnahme der Datenbank. Alte Revisionen der Dokumente werden erst bei einer vom Benutzer eingeleiteten *Compaction (Verdichtung)* gelöscht. Deshalb können Lesezugriffe ihr Ergebnis fertig abfragen, auch wenn gleichzeitig eine neue Version des Dokuments erstellt wird.

Wenn ein B+-Baum auf die Festplatte geschrieben wird, werden die Änderungen stets an das Ende der Datei angehängt. Dadurch wird die Zugriffszeit beim Schreiben auf die Festplatte minimiert. Außerdem wird verhindert, dass unvorhergesehene Beendigung des Prozesses oder Stromausfälle den Index korrumpieren:

> If a crash occurs while updating a view index, the incomplete index updates are simply lost and rebuilt incrementally from its previously committed state. [Apa09b]

5 Technische Grundlagen

In diesem Kapitel werden zunächst die einzelnen Webtechnologien dargestellt, die in die Anwendung eingeflossen sind. Anschließend wird auf die Hintergründe von Cloud Computing eingegangen, das zum Deployment der Anwendung verwendet wurde. Der letzte Abschnitt stellt die Methoden und Mittel dar, die zur Entwicklung der Anwendung unterstützend beigetragen haben.

5.1 Webtechnologien

Die Interaktionsmöglichkeiten mit der Oberfläche der zu erstellenden Anwendung fallen gering aus. Der Einsatz eines aufwändigen View Frameworks ist deshalb nicht notwendig; die Umsetzung kann mit den vergleichsweise einfachen Technologien, Frameworks und Programmiersprachen erfolgen, die hier im Einzelnen vorgestellt werden.

5.1.1 CouchApp

Die zu entwickelnde Anwendung wird als sogenannte CouchApp [Che10] umgesetzt. Das Projekt CouchApp stellt eine Reihe von unterstützenden Komponenten zur Verfügung, die die Entwicklung einer Standalone-Anwendung mit CouchDB erleichtern. Das Design sieht vor, dass jeder Benutzer eine eigene CouchDB-Instanz offline auf seinem Endgerät installiert hat, in der die Anwendung läuft. Dadurch ist die Verfügbarkeit auch ohne Internetanbindung gegeben.

Damien Katz, der Urheber von CouchDB, erklärt in [Kat10] CouchApps in folgenden Worten:

> CouchDB, being an HTTP server, can host applications directly, so you can write applications and forward the HTML, CSS, and JavaScript through CouchDB. When you point your browser at it, the browser comes alive and starts the JavaScript. It becomes interactive as you query and update the server and everything. When everything is served from CouchDB, that's a CouchApp and you can replicate it around, just like the data.

Eine CouchApp kann auf dem lokalen Rechner, einem Mobiltelefon, einem lokalen Server oder in der Cloud deployt sein, immer wird sie über einen Browser angesprochen. Durch Replikation können die Daten und auch das Programm immer wieder auf einen Stand gebracht werden.

Nach der Installation von CouchApp (erklärt in Abschnitt 9.1.2) kann das Gerüst für eine neue Beispiel-Applikation von der Kommandozeile mit dem Befehl `couchapp generate example-couchapp` erzeugt werden. Dies generiert ein Verzeichnis `example-couchapp`, das eine Verzeichnisstruktur vorgibt (s. Abb. 5.1). Dieses Verzeichnis entspricht einem CouchDB-Designdokument, wie es in Abschnitt 4.2.5 beschrieben ist.

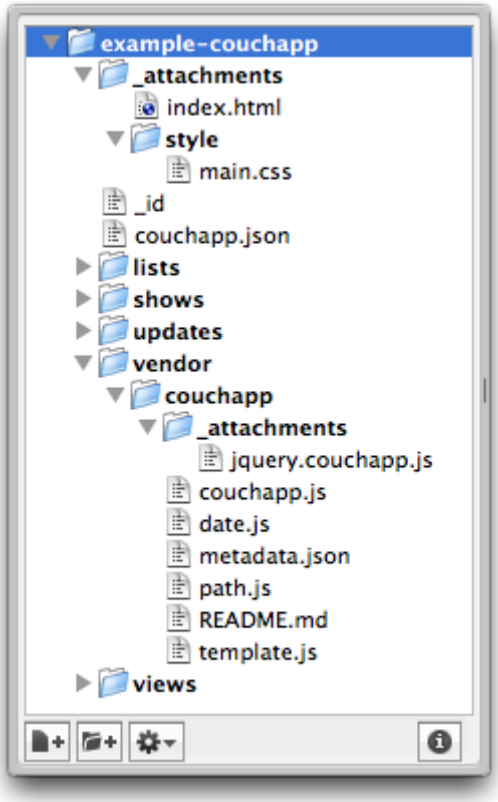

Abbildung 5.1: Generierte Beispiel-Couchapp

Das Verzeichnis `_attachments` enthält alle JavaScript-, HTML- und CSS-Dateien, die für Anwendungslogik und Darstellung benötigt werden. In der Beispiel-Anwendung ist eine einfache HTML-Startseite sowie ein Stylesheet vorgegeben. Die Verzeichnisse `lists`, `show`, `views` und `filters` enthalten entsprechend die List-, Show- bzw. Filter-Funktionen von CouchDB bzw. die Views. Im `vendor`-Verzeichnis werden externe Bibliotheken abgelegt, die für die Entwicklung benötigt werden.

Die Einstellungen für das Deployment werden in der Datei `.couchapprc` vorgenommen. Diese ist folgendermaßen formatiert:

```
1 {
2   "env": {
3     "default": {
```

```
4       "db": "http://user:password@localhost:5984/example-couchapp-dev"
5     },
6     "production": {
7       "db": "http:///user:password@example.com/example-couchapp"
8     }
9   }
10 }
```

Listing 5.1: Couchapp: .couchapprc

Dabei müssen für die Entwicklungs- und ggf. Produktionsumgebung Benutzername und Passwort des CouchDB-Administrators angegeben werden. Diese Information kann weggelassen werden, wenn sie nicht gesetzt sind. Mit dem Befehl `couchapp push` bzw. `couchapp push production` wird der Inhalt des CouchApp-Verzeichnisses in die CouchDB-Instanz kopiert.

Das URL-Schema einer CouchApp wird in Abschnitt 8.2 erläutert. In Abschnitt 9.1.2 wird erklärt, wie die in der Arbeit erstellte Anwendung mithilfe von CouchApp deployt wird.

5.1.2 HTML5

HTML (Hypertext Markup Language) ist das Hypertextformat im World Wide Web. *HTML5* [Hic10a] ist eine vom W3C entworfene Spezifikation, die in Zukunft die bisherigen HTML- und XHTML-Standards ersetzen soll. HTML5 wird mittlerweile von den meistbenutzten Browsern in den aktuellsten Versionen weitgehend unterstützt, mit der Ausnahme des Microsoft Internet Explorer [Smi09]. Diese Einschränkung verhindert bisher noch die Entwicklung der meisten Webanwendungen, da im Normalfall auf ältere oder nicht standardkonforme Browser Rücksicht genommen werden muss.

Wegen in Abschnitt 8.5.2 näher ausgeführten Einschränkungen ist die Anwendung auf den Browser Firefox ab der Version 3.6 festgelegt (siehe Release Notes [Moz10b]). Dies erlaubt es, auch bei zentralen Anwendungseigenschaften auf die Funktionen zuzugreifen, die HTML5 mit sich bringt. So bietet HTML5 neben vielen Verbesserungen die Möglichkeit, *Custom Data Attributes* zu definieren. Diese Technik wird in der Arbeit verwendet und wird deshalb kurz erklärt.

Laut Spezifikation [Hic10b] ist ein Custom Data Attribute ein Attribut ohne Namespace, dessen Name mit dem String `data-` beginnt und nach dem Bindestrich mindestens einen Buchstaben hat, der keine Großbuchstaben enthält. In Custom Data Attributes können eigene private Daten für die Seite oder die Anwendung gespeichert werden, wenn es keine passenden Attribute oder Elemente dafür gibt. Die Attribute sind dafür gedacht, von den eigenen Skripten der Seite benutzt zu werden, nicht als öffentlich nutzbare Metadaten. Jedes HTML-Element kann beliebig viele Custom Data Attributes haben.

5.1.3 JavaScript

JavaScript ist eine vielseitige Skriptsprache, deren Assoziation mit dem Webbrowser sie zu „einer der populärsten Programmiersprachen der Welt“ macht [Cro08, S. 2]. Über das *DOM (Document Object Model)* [W3C05] können mit JavaScript Objekte eines HTML-Dokuments direkt im Browser angesprochen werden. JavaScript ist eine dynamische, objektorientierte Programmiersprache, die ein prototypenorientiertes Paradigma verfolgt.

Am weitesten verbreitet ist der Einsatz von JavaScript zur Aufwertung der User Experience. Webseiten werden dabei mit clientseitigen Funktionalitäten „angereichert“, sind aber auch ohne JavaScript benutzbar. JavaScript kann ebenso als vollwertige Sprache serverseitig eingesetzt werden. Dabei wird Objektorientierung, anders als beispielsweise in Java oder C, durch Prototypen anstatt durch Klassen umgesetzt.

Im Folgenden werden der JavaScript-Bestandteil JSON, das Konzept AJAX sowie die Bibliothek jQuery vorgestellt.

5.1.3.1 JSON

JSON (JavaScript Object Notation) ist das populärste Format, um in JavaScript Informationen auszutauschen [Che07, Kap. 2]. JSON ist ein Subset von JavaScript [Cro06], es ist also selbst valides JavaScript. Nicht alle in JavaScript vorkommenden Datentypen können in JSON abgebildet werden, nur die Datentypen `Object`, `Array`, `String`, `Number`, `Boolean` und `Null` sind bekannt. Fast alle verbreiteten Programmiersprachen haben jedoch äquivalente Datentypen. So ist JSON auch zum Austausch mit anderen Sprachen gut geeignet [Cro10].

Beispiele von JSON finden sich in den Listings 5.5 und 8.12.

5.1.3.2 AJAX

AJAX steht für *Asynchronous JavaScript And XML*. Es bezeichnet kein Paket oder Framework. Jesse James Garrett, der den Begriff 2005 in [Gar05] prägte, beschreibt AJAX als

> [...] an approach — a way of thinking about the architecture of web applications using certain technologies.

Die Bestandteile von AJAX sind laut [Gar05] mehrere Technologien aus dem Bereich der Webentwicklung: die Repräsentierung durch XHTML und CSS, dynamische Interaktion durch das DOM, Datenaustausch und -manipulation durch XML und XSLT (oder ein anderes Datenaustauschformat wie JSON), asynchrone Datenanforderung durch XMLHttpRequest, und JavaScript für die

Verbindung der Komponenten. Mit diesem Setup wird es möglich, Daten zwischen Browser und Server asynchron zu übertragen.

Eine Webanwendung ist klassischerweise so aufgebaut, dass eine Aktion auf der Benutzeroberfläche eine HTTP-Anfrage zu einem Webserver auslöst. Der Server berechnet das Ergebnis mithilfe von Anwendungslogik und/oder Datenbankabfragen und schickt eine HTML-Seite an den Client zurück (vgl. [Garo5] und Abbildung 5.2).

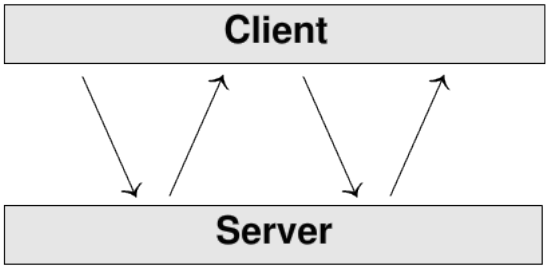

Abbildung 5.2: Synchrones Interaktionsschema einer traditionellen Webanwendung, nach [Rayo7]

Der Einsatz der oben aufgeführten Technologien erlaubt dem Browser, Daten vom Server asynchron im Hintergrund zu laden, ohne Darstellung und Verhalten der geöffneten Seite zu verändern. Schickt der Server eine Antwort auf die asynchrone Anfrage, werden nur die Teile der Seite verändert, für die neue Daten vorliegen - sie muss nicht neu geladen werden (Abbildung 5.3). Das hat den Vorteil, dass die Seite nicht ständig neu aufgebaut und weniger Daten übertragen werden müssen.

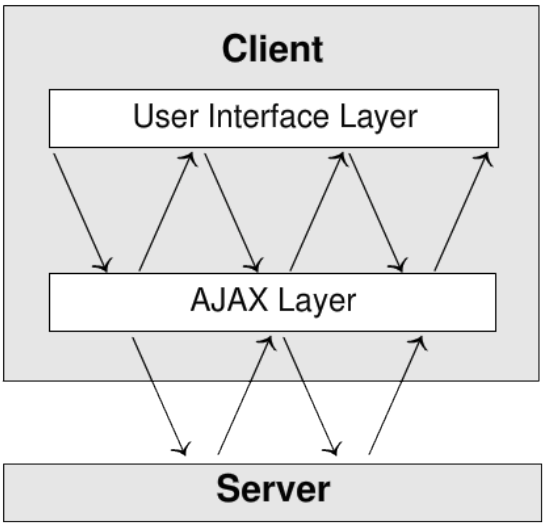

Abbildung 5.3: Asynchrones Interaktionsschema einer AJAX-Webanwendung, nach [Rayo7]

5.1.3.3 jQuery

Eine JavaScript-Bibliothek ist eine Sammlung vorgefertigter JavaScript-Funktionen, die die Entwicklung von Webanwendungen erleichtern. Die Benutzeroberfläche der Anwendung wird mit der JavaScript-Bibliothek jQuery [jQu10] in der Version 1.4 entwickelt:

> jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. [jQu10]

JQuery abstrahiert grundlegende JavaScript-Funktionalitäten. Insbesondere das Traversieren des DOMs wird dadurch stark vereinfacht. Dabei gibt jQuery keine Struktur für die Anwendung vor.

In einer nicht repräsentativen Umfrage, die im April 2010 von der Organisation *Web Directions* unter professionellen Webentwicklern und -designern durchgeführt wurde, gaben 78% der Befragten an, bei JavaScript-Entwicklung jQuery zu benutzen. [All10]

5.1.4 Sammy.js

Sammy.js [Qui10a] ist ein auf jQuery aufsetzendes Routing Framework für JavaScript-Anwendungen. Mit Sammy können *Routen* definiert werden, mit denen ein bestimmtes Verhalten verknüpft wird. Dadurch kann der Controller-Teil von Applikationen eventbasiert und RESTful umgesetzt werden. In Abschnitt 8.2 ist die Anwendung von Sammy in der implementierten Anwendung beschrieben.

5.1.4.1 Routen

Eine Sammy-Route besteht aus folgenden Bestandteilen:

Ein Pfad: Der *Anker*-Bestandteil einer URL, also der Teil nach dem URL-Hash (#). Er kann in der Spezifikation der Route als String oder als Regulärer Ausdruck angegeben werden.

Eine Methode: Eine der HTML-Methoden `GET`, `POST`, `PUT` oder `DELETE`.

Ein Callback: Eine Funktion, die beim Aufrufen des Pfades mit der Methode aufgerufen wird.

Auch wenn der Pfad einer Route derselbe ist, können je nach Methode unterschiedliche Callbacks gesetzt werden. So kann z. B. unter der Route „`get('#/outlines/:id')`“ die Ressource Outline angezeigt und unter „`put('#/outlines/:id')`“ aktualisiert werden. Durch die Benutzung des URL-Ankers können clientseitige Anwendungen auf einer einzigen Seite umgesetzt werden, die trotzdem auf den „Zurück“-Button des Browsers reagieren.

Es ist Aufgabe der Sammy-Bibliothek, benannte Parameter aus dem Pfad herauszuparsen. Dadurch können IDs oder Slugs aus dem Pfad herausgefiltert werden. Jeder String innerhalb des Pfades, der mit dem Zeichen „:“ startet, wird in einen benannten Parameter umbenannt.

Routen mit den Methoden `POST`, `PUT` und `DELETE` werden nur von abgeschickten HTML-Formularen aufgerufen. Zur Laufzeit wird die `submit`-Methode für alle Formulare überschrieben und an Sammy gebunden. Beim Abschicken des Formulars wird nach einer Route gesucht, die dem Pfad des Formulars und der Methode entspricht. Wird eine solche Route gefunden, wird ihr Callback ausgeführt.

Routen können auch an benutzerdefinierte Events gebunden werden, die dann von der Anwendung ausgelöst werden. So kann beispielsweise beim ersten Aufrufen der Sammy-Anwendung eine `init`-Funktion aufgerufen werden, die wiederum Elementen auf der Seite bestimmtes Verhalten zuweist.

5.1.4.2 Sammy-Plugins

Es besteht die Möglichkeit, für Sammy eigene Plugins zu definieren. Ein Plugin ist Programmcode, der wie der Rest der Bibliothek eingebunden, aber erst auf Anforderung hin ausgewertet wird.

Das folgende Beispiel, entnommen [Qui10b], definiert die Helper-Funktion `alert()`, die die JavaScript-Funktion `alert` überschreibt und durch einen Eintrag im Logfile ersetzt:

```
1 var MyPlugin = function(app) {
2   this.helpers({
3     alert: function(message) {
4       this.log("ALERT! " + message);
5     }
6   });
7 };
```

Listing 5.2: Sammy.js: Beispiel für ein Plugin

Ein Plugin wird mit der Methode `use()` aufgerufen. Dadurch wird die Plugin-Funktion innerhalb des Kontexts der aktuellen Sammy-Applikation ausgewertet. Wenn das Beispiel-Plugin aktiviert ist, kann die Methode innerhalb aller Routen benutzt werden.

```
1 var app = $.sammy(function() {
2   this.use(MyPlugin);
3   this.get('#/', function() {
4     this.alert("I'm home"); //=> logs: ALERT! I'm home
5   });
6 });
```

Listing 5.3: Sammy.js: Einbinden des Plugins

5.1.5 Mustache.js

Für das Rendern der HTML-Seiten wird das Template Engine *Mustache* [Leh10c] eingesetzt. Ein Template Engine ist eine Software, die in einer Datei bestimmte Platzhalter mit mitgegebenen Inhalten füllt. Mustache-Implementierungen existieren in vielen Sprachen, hier wird die JavaScript-Version *Mustache.js* verwendet.

Durch den Einsatz von Mustache kann die Trennung von Logik und Repräsentierung umgesetzt werden [Leh10b]. Die Repräsentierung wird in einer HTML-Datei vorgenommen, die für nicht-statische Werte Platzhalter enthält. Die Programmlogik, also die Berechnung der ausgegebenen Werte oder die einfache Deklaration der Variablen, wird in einer *View* platziert. Eine View ist ein JSON-Objekt mit Attributen und Methoden, die mit den Platzhaltern im Template korrespondieren.

Das folgende Beispiel demonstriert dies (nach [mus10]):

```
1 Hallo {{name}},
2 du hast gerade {{brutto_wert}}$ gewonnen!
3 {{#steuerpflichtig}}
4 Also, {{netto_wert}}$, nach Steuern.
5 {{/steuerpflichtig}}
```

Listing 5.4: Mustache.js: Beispiel für ein Template

```
1 {
2   "name": "Chris",
3   "brutto_wert": 10000,
4   "netto_wert": 10000 - (10000 * 0.4),
5   "steuerpflichtig": true
6 }
```

Listing 5.5: Mustache.js: Übergebene View

Die Mustache-Bibliothek ist eine JavaScript-Datei, die zur Laufzeit geladen werden muss. Durch den Aufruf der Methode `Mustache.to_html(template, view)` werden das Template und die View gerendert:

```
1 Hallo Chris,
2 du hast gerade 10000$ gewonnen!
3 Also, 6000.0$, nach Steuern.
```

Listing 5.6: Mustache.js: Ergebnis

Da bei Mustache keine Logik in den Templates umgesetzt werden muss, wird Programmieren nach dem *MVC*-Architekturmuster *(Model-View-Controller)* unterstützt [Leh10b]. Bei einem ak-

tuellen von Brian Landau durchgeführten Benchmarking Vergleich mit sieben anderen JavaScript Templating Libraries schnitt Mustache.js sehr gut ab [Lan09a].

5.1.6 Weitere Bibliotheken

Die Gestaltung der Oberfläche wird mit HTML und *Cascading Style Sheets (CSS)* vorgenommen. Für die Seitenaufteilung wird das CSS Framework *Blueprint* [Mon] verwendet. Mit Blueprint wird ein Container mit einer bestimmten Pixelbreite erzeugt, in dem ein rasterbasiertes Layout umgesetzt werden kann. Dieses Raster ist in 24 Spalten unterteilt. Die Verteilung der Elemente auf die Seite wird vorgenommen, indem den Elementen ein bestimmtes `class`-Attribut zugewiesen wird. Ein `div`-Element mit der Auszeichnung `<div class="column span-16">` füllt automatisch zwei Drittel des Containers aus. Blueprint stellt weiterhin *Cross-Browser-Kompatibilität* sicher; eine damit umgesetzte Webseite verhält sich unabhängig vom Browsertyp weitestgehend identisch. Außerdem enthält Blueprint mehrere Stylesheets, die Voreinstellungen für ein Design liefern können.

Das jQuery-Plugin *jquery.autogrow* [Bad08] erlaubt es, die Größe der Textareas, die die Zeilen des Gliederungseditors repräsentieren, automatisch an die Menge des enthaltenen Textes anzupassen. Während Text eingegeben wird, wächst die Textarea in der Breite und ggf. der Länge mit. Dies wird erreicht, in dem auf dem DOM-Element die entsprechende Funktion aufgerufen wird: `$('textarea').autogrow();`

Des Weiteren wurden die jQuery-Plugins *jquery.md5*, *jquery.unwrap*, *jquery.scrollTo*, *jquery.color* und *date.format* verwendet.

5.2 Cloud Computing

Das Deployment der Anwendung wurde mit dem sogenannten *Cloud Computing* vorgenommen. Der Begriff wird als eine Metapher für über das Internet angebotene Dienste verwendet, da dieses in Computernetzwerkdiagrammen häufig als Wolke dargestellt wird (s. Abb. 5.4).

Cloud Computing wurde 2006 von dem CEO von Google in einem Talk über „Search Engine Strategies“ eingeführt [Sch06]. Der Begriff ist schwer eindeutig zu definieren, in [Qia09, S. 626] wird es als „eine der vagesten Terminologien in der Technik-Geschichte“ bezeichnet. Ein Grund dafür ist, dass Cloud Computing für viel Verschiedenes eingesetzt wird. Außerdem wird der Begriff von vielen Firmen für Business-Reklame benutzt. Im Jahr 2009 war er eines der aktuellen Schlagworte der IT-Branche, das am häufigsten zur Erzeugung überhöhter Erwartungen benutzt wurde:

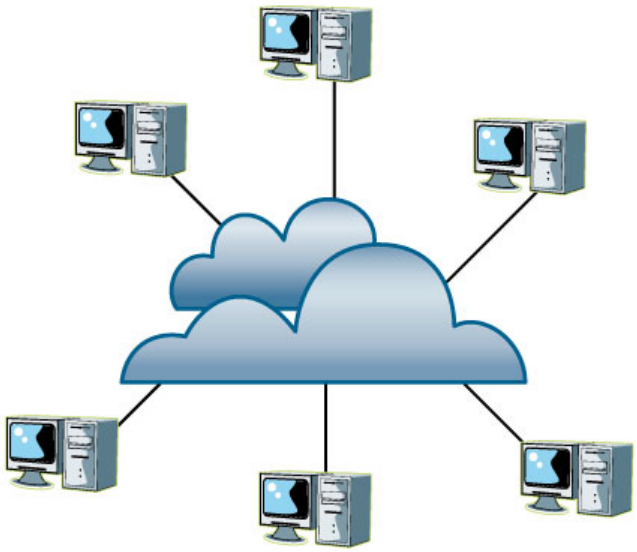

Abbildung 5.4: Cloud Computing: Eine Metapher für über das Internet angebotene Dienste [P. 09]

> The levels of hype around cloud computing in the IT industry are deafening, with every vendor expounding its cloud strategy and variations [...], compounding the hype. [Gar09]

Laut des zitierten Berichts wird Cloud Computing in weniger als fünf Jahren im „Mainstream“ angekommen sein (s. Abb. in Abschnitt A.2.2).

Auch wenn keine einheitliche Definition des Begriffes Cloud Computing besteht, lässt sich trotzdem eine Einigkeit über grundlegende Konzepte und generelle Ziele feststellen. Verschiedene Definitionen werden im folgenden Abschnitt ausgeführt. Danach werden die gebräuchlichen Stile von Cloud Computing dargestellt, und im letzten Abschnitt werden Vor- und Nachteile für den Einsatz abgewogen.

5.2.1 Definition

Zunächst werden zwei Definitionen aus der Fachliteratur genannt:

> Cloud Computing erlaubt die Bereitstellung und Nutzung von IT-Infrastruktur, von Plattformen und Anwendungen aller Art als im Web elektronisch verfügbare Dienste. [Bau10, Kap. 1.1]

> Cloud Computing is a kind of computing technique where IT services are provided by massive low-cost computing units connected by IP networks. [Qia09, S. 627]

[Qia09] nennt als weitere zentrale Charakteristika von Cloud Computing die Virtualisierung der angebotenen Dienste, das dynamische Scheduling der Ressourcen und eine hohe Skalierbarkeit. Benötigt eine Anwendung zusätzliche Ressourcen, können diese sofort und ohne großen Aufwand automatisch dazu geschaltet werden. Die Infrastruktur passt sich automatisch den schwankenden oder wachsenden Anforderungen an.

[Bau10] betont als Kriterium auch, dass die Abrechnung dieser Cloud-Computing-Dienste üblicherweise nutzungsabhängig erfolgt: Es wird wird immer die aktuell benötigte Menge an Ressourcen zur Verfügung gestellt und bezahlt. Signifikante Kostenersparnisse sind aufgrund der flexiblen Bereitstellung und Nutzung von Diensten möglich.

Die oben genannten Definitionen legen nicht fest, ob Dienste auf Basis eines verteilten Systems oder eines einzelnen leistungsstarken Servers erbracht werden. Das steht im Gegensatz zum Begriff *Grid Computing*, der immer Verteilte Systeme bezeichnet.

5.2.2 Stile

Cloud Computing entwickelte sich, als um den Jahrhundertwechsel die Expansion des Internets einen großen Druck auf die existierenden Speicher- und Recheneinrichtungen ausübte. Personal Computer wurden als Ressource immer billiger; Internet Service Provider begannen, diese als die zugrundeliegende Hardware-Plattform zu nutzen [Qia09]. Um Computer-Cluster flexibel einzusetzen wurden verschiedene Softwaremodelle entwickelt. So konnten Rechenressourcen abstrahiert werden. Daraus entwickelten sich drei größere Cloud Computing Stile. Die Vorstellung dieser Stile bezieht sich auf [Qia09].

Das Cloud Computing-Konzept der Firma Amazon basiert auf der Technologie der *Server-Virtualisierung*. Unter dem Namen *Amazon Web Services (AWS)* werden seit 2006 mehrere Webservices angeboten. Diese bieten virtualisierte Rechenressourcen zur generischen Nutzung an. Da AWS bei seiner Einführung günstiger war als vorherige Methoden zur „on-demand"-Bereitstellung dieser Services, wurde es zum Pionier der „Infrastructure as a Service (IaaS)" Anbieter. In dieser Diplomarbeit werden das auf der Virtualisierungssoftware *Xen* basierende *Elastic Compute Cloud (EC2)*, der *Simple Storage Service (S3)* und den Speicher *Elastic Block Store (EBS)* verwendet. All diese Services sind Bestandteile von AWS.

Bekannte Vertreter der beiden weiteren Cloud Computing-Stile sind u.a. die Firmen Google und Microsoft. Google bietet *technique-specific sandboxes* an, die das einfache Hosting von mit bestimmten Technologien umgesetzten Anwendungen erlauben. Diese Technik wird auch bei der Firma Heroku für mit der Programmiersprache Ruby erstellte Anwendungen eingesetzt. Microsoft

bietet mit dem Service *Azure* eine Kombination aus Server-Virtualisierung und technique-specific Sandboxes.

Server-Virtualisierung gilt als am flexibelsten und kompatibelsten mit existierender Software und Anwendungen. Die anderen beiden Ansätze führen zu höheren Einschränkungen bei der Wahl der Programmiersprache, da jeder Service nur bestimmte Technologien unterstützt. Dafür ist bei Server-Virtualisierung der Abstraktionsmehraufwand höher. Dieser Ansatz ist im Moment die populärste Technik im Cloud Computing, Dienste und Ressourcen zu abstrahieren.

Mit diesen unterschiedlichen Stilen lassen sich verschiedene Arten von Cloud Computing umsetzen. Diese werden nach [Eymo8] in drei Ebenen eingeteilt:

Software as a Service (SaaS) - Ein SaaS-Provider bietet eine Software im Internet als Dienst. Dieser kann in Anspruch genommen werden, ohne dass der Benutzer Kenntnis oder Kontrolle über die dem Service zugrundeliegende Infrastruktur hat.

Plattform as a Service (PaaS) - Ein Paas-Provider stellt eine Plattform zur Verfügung, durch die ein leichterer Zugang zu einer Kombination aus unterschiedlichen Services ermöglicht wird.

Infrastruktur as a Service (IaaS) - Von einem IaaS-Provider wird Hardware als Infrastrukturdienst angeboten. Auf diesem können dann eigene Services betrieben werden.

In Abbildung 5.5 sind für die drei Ebenen Zielgruppen und Beispiele für Anbieter aufgeführt.

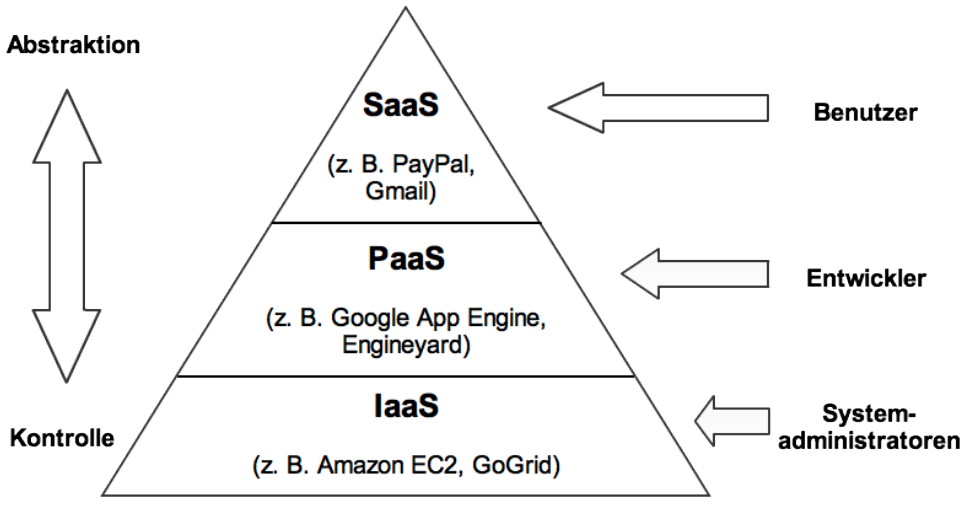

Abbildung 5.5: Darstellung der drei Ebenen von Cloud Computing

5.2.3 Vor- und Nachteile

Cloud Computing wird in [Qia09, S. 629] als eine „Win-win-Strategie“ für sowohl Anbieter als auch Benutzer der Services beschrieben.

Als vorteilhaft wäre zu nennen, dass Geschäftsanforderungen „on-demand“ erfüllt werden können: Kunden können über die Anwendung die Größe der benutzten Ressourcen flexibel einstellen, um sie an die tatsächlichen Anforderungen anzupassen. Dadurch ergeben sich die weiteren Vorteile, dass Kosten und Energieverbrauch gesenkt werden können. Das Ressourcenmanagement kann ebenfalls durch dynamisches *Resource-Scheduling* verbessert werden.

Die Nachteile beinhalten, dass bei der Beanspruchung der Dienste von traditionellen Hosting-Anbietern die Fragen bezüglich Datenschutz und Sicherheit besser geklärt sind, da hier erst einmal nur der Benutzer Kenntnisse über die Beschaffenheit des aufgesetzten Systems hat. Des Weiteren ist die Verlässlichkeit der Dienste bei technischen Defekten, Stromausfällen o.Ä. nicht gewährleistet. Dies hat bei Cloud Computing u.U. gravierendere Auswirkungen, da solch ein Zwischenfall nicht nur einen einzelnen Server, sondern alle Dienste auf einmal ausschalten können. Derartige Unsicherheiten können mit *Service Level Agreements* abgefangen werden, in denen Leistungsqualität und -umfang geregelt werden. Bedenken gegenüber Cloud Computing kann ebenfalls angebracht sein, weil das Migrieren der Dienste zu einem anderen Anbieter meistens unmöglich ist. Bisher sind externe Schnittstellen kaum definiert. Aktuelle Angebote sind wegen eines Wettbewerbsvorteils meist proprietär [Bau10].

Um die finanziellen Vorteile von Cloud Computing abschätzen zu können, müssen die Kosten der tatsächlichen Nutzung von Cloud-Angeboten in Stunden oder Server-Einheiten den Kosten des Rechenzentrums oder der angeschafften IT-Infrastruktur entgegengestellt werden. Dabei sollte die durchschnittliche Auslastung des Rechenzentrums mit einbezogen werden. Beachtet werden muss aber, dass ein Rechenzentrum fixe Kapazitäten hat, Cloud-Dienste dagegen über eine nach oben offene Kapazitätsgrenze verfügen [Bau10, Kap. 7.2.1].

5.3 Methoden und Mittel

Nachdem in den vorangehenden Abschnitten alle Technologien behandelt wurden, die direkt in die fertige Anwendung eingeflossen sind, werden im Folgenden die für die Umsetzung verwendeten Werkzeuge vorgestellt. Dieses Kapitel enthält also all die Technologien und Methoden, die unterstützend zum Einsatz kamen, die aber nicht Teil des Endprodukts sind.

Zu Beginn werden die beiden verwendeten Vorgehensmodelle, namentlich die Agile Softwareentwicklung und die Testgetriebene Enwicklung, dargestellt. Danach werden die Testing Frameworks

beschrieben, mit deren Hilfe die Testgetriebene Entwicklung umgesetzt wird. Ein Abschnitt über die Entwicklungsumgebungen rundet das Kapitel ab.

5.3.1 Vorgehensmodelle

Als Vorgehensmodell für die Entwicklung wird der Ansatz der Agilen Softwareentwicklung gewählt. Testgetriebene Entwicklung kann als eine Untermenge der in Agiler Softwareentwicklung enthaltenen Vorgehensweisen verstanden werden. In dieser Arbeit wird der Testgetriebenen Entwicklung allerdings ein hoher Stellenwert eingeräumt, deswegen wird ihr ein eigener Abschnitt gewidmet.

5.3.1.1 Agile Softwareentwicklung

Im Bereich der Softwareentwicklung wuchs bereits gegen Ende der achtziger Jahre die von verschiedenen Seiten geäußerte Kritik an den herkömmlichen Phasen- und Vorgehensmodellen. [Hes92] nennt Quellen eines weitverbreiteten Unbehagens über das klassische „life cycle"-Konzept. Der Autor führt das Unbehagen über klassische Modelle nicht nur auf einen Modetrend zurück, es „beruht auch auf ernstzunehmenden Erfahrungen mit den herkömmlichen Modellen und dabei festgestellten Schwächen" [Hes92, Kap. 2.5.1]. Diese beinhalten neben zu langen Zeiträumen zwischen Spezifikation und lauffähigem Programm und ungenügender Miteinbeziehung von Kunden und Anwendern vor allem den mangelnden Realitätsbezug der streng aufeinanderfolgenden Phasen.

2001 legten einige namhafte Vertreter der Agilen Softwareentwicklung deren Grundwerte im sogenannten *Agilen Manifest* fest [Bec01]. Diese Werte bilden das Fundament des mit diesem Manifest erstmals genauer definierten Entwicklungsprozesses. Angewendet auf den Entwicklungsprozess der hier umgesetzten Anwendung ergeben sich aus den Werten folgende Ziele:

- Häufige Rückkopplung und Kommunikation zwischen allen Projektbeteiligten
- Frühe und häufige Softwareauslieferungen; dadurch kann überprüft werden, ob der Entwicklungsprozess auf „dem richtigen Weg" ist, das eigentliche Ziel hinter dem Projekt zu erreichen
- Die Möglichkeit, die anfänglich festgelegten Pläne bezüglich Anforderungen und Verfahren den tatsächlichen Anforderungen anzupassen

Aufbauend auf den Grundwerten des Agilen Manifests definiert [Amb] Agile Softwareentwicklung wie folgt:

> Disciplined agile software development is: an iterative and incremental (evolutionary) approach to software development; which is performed in a highly collaborative manner; by self-organizing teams within an effective governance framework; with „just enough“ ceremony; that produces high quality software; in a cost effective and timely manner; which meets the changing needs of its stakeholders.

Bei der Entwicklung dieser Arbeit wird nach Agilen Methoden vorgegangen. Darunter wird eine etablierte Handlungsweise verstanden, in einem ausgewählten Ausschnitt oder Aspekt der Softwareentwicklung agil vorzugehen [Bleo8, Kap. 2.4]. Insbesondere ständiges Code Refactoring, Continuous Integration und die Testgetriebene Entwicklung wären hier zu nennen.

Code Refactoring wird in [Hamo8] definiert als *Behaviour-preserving Transformation*. Refactoring ist demnach „der Prozess, Quelltext zu transformieren um sein internes Design zu verbessern, ohne seine externe Funktionalität zu verändern“ [Hamo8, S. 2]. Bei Continuous Integration werden alle Tests immer dann automatisiert ausgeführt wenn neuer Code in die Anwendung integriert wird. So wird die kontinuierliche Funktionsfähigkeit der Anwendung sichergestellt. Testgetriebene Entwicklung wird im nächsten Abschnitt vorgestellt.

5.3.1.2 Testgetriebene Entwicklung

Die Anwendung wird testgetrieben entwickelt. Testgetriebene Entwicklung (*Test Driven Development, TDD*) ist eine der bedeutendsten und weitest verbreiteten Praktiken Agiler Softwareentwicklung [Hamo8, S. 2]. Mit ihr sollen hohe Qualität und gute Wartbarkeit des Programms sichergestellt werden.

In [Hamo8] wird der *Testgetriebene Entwicklungszyklus* beschrieben. TDD besteht aus drei immer wiederkehrenden Schritten: *Test - Code - Refactor*.

Test Der Test für den neu zu erstellenden Code wird geschrieben und gestartet. Da der Code noch nicht existiert bzw. das gewünschte Verhalten noch nicht implementiert ist, wird der Test fehlschlagen. Wichtig ist dabei, kleinschrittig vorzugehen, also nur einen Aspekt des Codes zu testen.

Code Der Code für das neue Feature wird geschrieben, in der denkbar einfachsten Implementierung. Der Test läuft nun erfolgreich durch.

Refactor Der Code wird durch Refactoring verbessert. Dabei muss beachtet werden, dass sämtliche Tests erfolgreich durchlaufen.

Dieses Verfahren wird auch als *test first programming* bezeichnet.

[Fiso8] nennt die Qualitäten eines guten Tests. Ein guter, d.h. sinnvoller Test muss **aussagekräftig** sein, also ein genau definiertes Ja/Nein-Ergebnis liefern. Er muss weiterhin **gültig** sein, das

Testresultat muss der Intention des getesteten Artefakts entsprechen. Man spricht von einem **kompletten** Test, wenn er zum Laufen keinen weiteren Input benötigt, und von einem **wiederholbaren** Test, wenn das Resultat deterministisch ist, auch wenn das getestete System sich nicht deterministisch verhält. Ein Test sollte weiter völlig **isoliert** sein, das Resultat darf demnach nicht durch Resultate oder Seiteneffekte eines anderen Tests beeinflusst werden. Dieses Anti-Pattern wird in [Ham08] als *Test-Coupling* bezeichnet und sollte vermieden werden. Als letzte Eigenschaft eines guten Tests wird gefordert, dass dieser **automatisiert** angestoßen werden können muss, in **endlicher** Zeit fertig sein und mit anderen Tests in einer Testsuite **zusammengefasst** werden können soll.

Eine Weiterentwicklung von TDD ist *Behaviour Driven Development (BDD)*. Hier wird die Betonung vom Aspekt des „Testens“ hin zum Aspekt der „Vorab-Spezifikation“ verschoben [Ada07, Kap. 1]. Dabei wird sich dem Ergebnis, ähnlich wie beim Domain Driven Design, von der Geschäftsseite genähert. So kann die Sprache, mit der das zu lösende Problem beschrieben wird, weitgehend frei von technischen Fachbegriffen gehalten werden. Für BDD existieren zahlreiche Frameworks, die es erlauben, die Spezifikationen für eine Software in ausführbarem Code auszudrücken:

> A waterfall „designer“ starts from an understanding of the problem and builds up some kind of model for a solution, which they then pass on to the implementers. An agile developer does exactly the same, but the language they use for the model happens to be executable source code rather than documents or UML. [Ada07, Kap. 2]

Die Begriffe TDD und BDD werden in der Praxis oft mit gleicher Bedeutung verwendet [Mil07]. Entgegen von manchen Entwicklern geäußerter Bedenken führen TDD/BDD nicht zu mehr Aufwand und nicht zu längerer Entwicklungszeit. Je früher und je umfassender Tests entstehen, desto schneller und problemloser kann der Entwicklungszyklus ablaufen. Deshalb soll die Anwendung mit dieser Vorgehensweise implementiert werden.

5.3.2 Testing Frameworks

Die während der Testgetriebenen Entwicklung geschriebenen Tests lassen sich in unterschiedliche Ebenen aufteilen. *Unit Tests* prüfen die Funktionalität der einzelnen Software-Module. *Integration Tests* dagegen prüfen das Zusammenspiel aller Komponenten des Systems. Es gibt in Abhängigkeit der benutzten Frameworks weitere Ebenen. Für diese Arbeit erfolgte jedoch eine Beschränkung auf diese beiden, da sie in Kombination und bei guter Umsetzung eine sehr gute funktionale Testabdeckung bieten. Im Folgenden werden die jeweils eingesetzten Frameworks vorgestellt.

5.3.2.1 JSpec

Ein *Unit Test Framework* ist eine Software, mit der das Schreiben und das Ausführen von Unit Tests unterstützt wird. Solche Frameworks liefern eine Grundlage, die die Erstellung von Tests erleichtert, sowie Funktionalität um die Tests auszuführen und die Ergebnisse auszugeben. Unit Tests werden neben der eigentlichen Anwendung entwickelt, sie sind nicht in das endgültige Software-Produkt eingebunden. Sie benutzen die Objekte der Anwendung, existieren aber nur innerhalb des Unit Test Frameworks. So können Objekte isoliert voneinander getestet werden, und greifen nicht in den eigentlichen Code ein [Ham08].

An dieser Stelle kann aufgrund des begrenzten Umfangs dieser Arbeit kein fundierter Vergleich der in Frage kommenden Unit Test Frameworks vorgenommen werden. Eine von der Autorin vorgenommene Evaluation findet sich in [Her10]. Unter mehreren respektablen Alternativen [Wik10b] fiel die Wahl auf das relativ junge Framework JSpec [Hol10].

JSpec ist in Funktionalität und Syntax an das BDD Framework *RSpec* [Che] angelehnt, mit dem Ruby-Code spezifiert und getestet werden kann. Die JSpec-Syntax ist eine eigens hierfür entwickelte *domänenspezifische Sprache (DSL)*.

Matcher spezifizieren bestimmtes Verhalten oder den Wert eines Objekts. *Assertions/Expectations* überprüfen dies bzw. vergleichen es mit einem bestimmten Wert. Das Schlüsselwort `should` ist mit dem Schlüsselwort `assert` aus herkömmlichen *xUnit Test Frameworks* zu vergleichen, die auf dem *SUnit Framework* von Kent Beck basieren.

```
1 { foo : 'bar' }.should.eql { foo : 'bar' }
```

Listing 5.7: JSpec Beispiel: Der Matcher `eql`

Weitere Beispiele finden sich in Anhang A.5.8.2.

Es existieren ebenfalls Matcher für JQuery-Funktionalität, mit denen das DOM getestet werden kann. JSpec bietet außerdem Unterstützung für das Testen von asynchronen Funktionen und die Möglichkeit, AJAX-Requests zu simulieren. In *Fixtures* können Teile des DOMs als HTML-Code bereitgestellt werden.

JSpec kann entweder als Ruby-Gem installiert werden, dann können die Tests von der Konsole aus aufgerufen werden und so auch in Continuous Integration integriert werden. Dabei wird auf Rhino [Mozc], einen auf Java basierenden JavaScript-Interpreter, zurückgegriffen. In der Konsole kann angegeben werden, welcher Browser im Hintergrund geöffnet werden soll; dieser Browser wird dann die Tests ausführen.

Alternativ kann die JSpec-Bibliothek in den JavaScript-Code eingebunden werden. Bei dieser Vorgehensweise wird zum Ausführen der Tests eine HTML-Datei im Browser geöffnet, die im selben

Browser die Tests ausführt. In diesem Fenster werden dann auch die Ergebnisse angezeigt. Für Detailliertheitsgrad und Formatierung der Ergebnisse können in der HTML-Datei verschiedene Parameter angegeben werden. Ein Beispiel für die Anzeige findet sich in Abbildung 5.6.

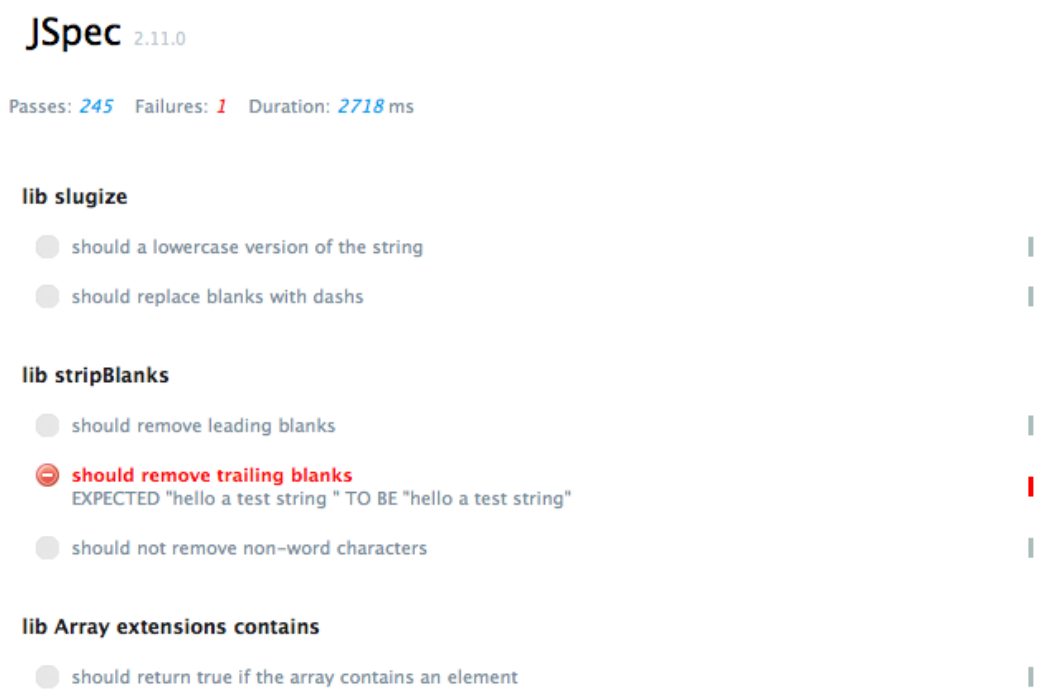

Abbildung 5.6: JSpec: Ein Test schlägt fehl

5.3.2.2 Cucumber

Bei TDD werden Tests vor dem Programmcode geschrieben, deshalb handelt es sich um Black-Box-Tests [Bec07]. Bei diesen ist die Implementierung der zu testenden Programmkomponente nicht bekannt, es wird ausschließlich die zu erwartende Funktionalität bzw. das Ergebnis getestet. Bei Unit Tests ist dies oftmals nicht strikt umsetzbar, da diese sehr feinkörnig Eigenheiten der Komponente testen. Integration Tests dagegen setzen auf einer höheren Ebene an: Hier wird die Funktionalität einer Komponente so beschrieben, dass auch Benutzer ohne technischen Hintergrund die Spezifikation verstehen können. Der Entwicklerin erlaubt dies, vor dem Schreiben des Programmcodes unbelastet von Implementierungsdetails über die Geschäftslogik nachzudenken.

Auch wenn eine Software-Komponente erfolgreich durch Unit Tests getestet ist, kann die Qualität erst dann als garantiert gelten, wenn die Komponente erfolgreich mit dem Rest der Anwendung integriert ist. Das Testing Framework Cucumber [Hel] bietet ein Grundgerüst, um solche Integration Tests zu erstellen.

Der Test für eine größere zusammengehörende Programmkomponente (z. B. Benutzerverwaltung, Benutzung des Gliederungseditors) wird in der *Domain Specific Language (DSL)* von Cucumber ein *Feature* genannt. Jedes Feature spezifiert die Rolle des Benutzers (*As a*), den Inhalt des Features (*I want*) sowie seinen Geschäftswert (*In order to*). Ein Feature enthält mehrere *Stories*, die jeweils die Ausführung einer Software-Funktion von Anfang bis Ende beschreiben (z. B. Benutzer-Login, Zeile einrücken). Eine Story enthält die Vorbedingungen (*Given*), die einzelnen Schritte des Benutzers (*When*) und die erwünschten Resultate (*Then*). Ein dem Projekt entnommenes Beispiel findet sich in Listing 5.8.

```
Feature: CRUD for outlines
  In order to sort my notes
  As a user
  I want to create, list, update and delete outlines

  Scenario: create an outline with note
    When I go to the start page
      And I follow "New Outline"
      And I fill in "title" with "Songs"
      And I press "Save"
    Then I should see "Songs"
      And I should see "Here is your new outline"
      And the new note li should be blank

  Scenario: edit an outlines title
    Given an outline with the title "Songs"
      And I save
    When I go to the start page
      And I follow "Songs"
      And I follow "Change title or delete this outline"
      And I fill in "title" with "Tunes"
      And I press "Save"
    Then I should see "Title successfully changed"
      When I go to the start page
    Then I should see "Tunes"
      And I should not see "Songs"
```

Listing 5.8: Ein Cucumber Feature mit zwei Szenarien

Die Bedeutung der einzelnen Zeilen, *Steps* genannt, muss in weiteren Dateien definiert werden. Jeder Step besteht aus einem Signalwort und einem Regulären Ausdruck, für den ein Block mit Ruby-Code ausgeführt wird. Dabei werden die Resultate der Matching-Gruppen im Regulären Ausdruck dem Block übergeben. Dies ist beispielhaft in Listing 5.9 demonstriert.

Ein Feature kann auf der Kommandozeile ausgeführt werden (Beispiel in Abbildung 5.7). Dabei werden die erfolgreichen Steps grün und die fehlgeschlagenen rot dargestellt, wobei bei letzteren die Fehlermeldungen mit ausgegeben werden.

```
Given /^an outline with the title "([^\"]*)"$/ do |title|
  outline = {:kind => 'Outline', :title => title}
  RestClient.put "#{host}/#{database}/#{title}", outline.to_json
end

When /I fill in "(.*)" with "(.*)"/ do |field, value|
  find_by_label_or_id(:text_field, field).set value
end
```

Listing 5.9: Cucumber Step-Definition

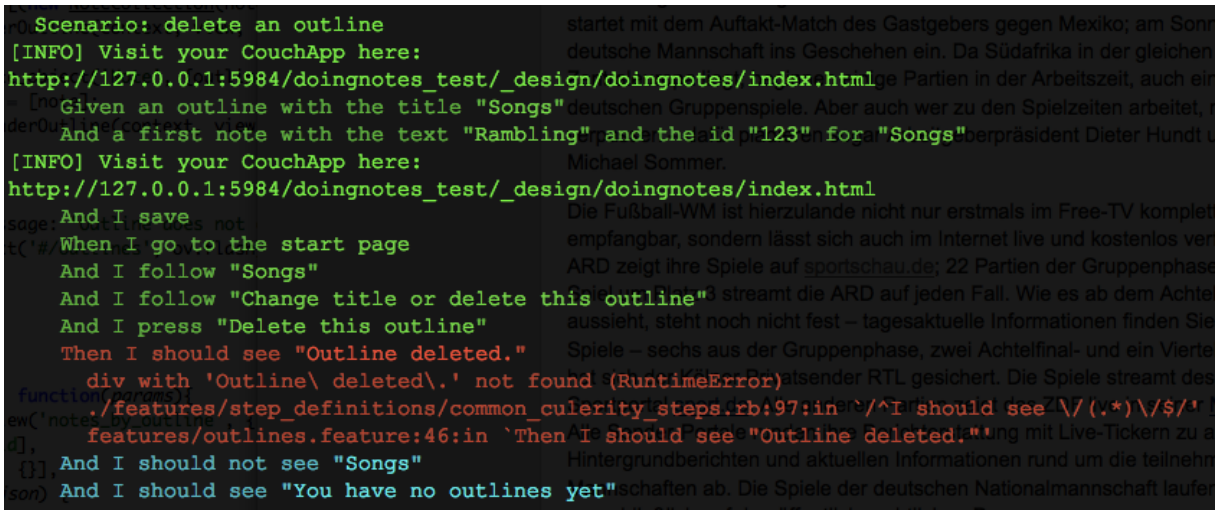

Abbildung 5.7: Cucumber: Ein Test schlägt fehl

5.3.3 Entwicklungsumgebungen

5.3.3.1 Textmate

TextMate [Mac10] ist ein Texteditor für das Betriebssystem Mac OS X. TextMate wurde 2004 von Allan Odgaard herausgegeben. Im August 2006 bekam das Programm den „Apple Design Award for Best Developer Tool“ bei der „Worldwide Developers Conference“ der Firma Apple. TextMate ist ein sehr übersichtlicher Editor [Ski07]. Er hat bei weitem nicht den Funktionsumfang einer Entwicklungsumgebung wie Eclipse oder NetBeans, lässt sich aber durch seine weitreichende Unterstüzung für Skripte und Plugins beliebig erweitern. Für die Bearbeitung der vorliegenden Aufgabe wäre der Einsatz einer solchen integrierten Entwicklungsumgebung nicht zweckmäßig gewesen: Die spezifischen Anforderungen, die die Entwicklung einer Couchapp stellt, werden durch Eclipse zum jetzigen Zeitpunkt nicht erfüllt.

Textmate bietet Syntax-Highlighting für alle verwendeten Sprachen, Auto-Completion innerhalb einer Datei, projektweite Suche, einfachen Zugriff auf alle Dateien im Projekt, ein übersichtliches Interface mit Tabs für alle geöffneten Dateien. Darüberhinaus ist es leicht, Textmate an besondere Anforderungen anzupassen. So wurde für das Aktualisieren der Designdokumente in der Datenbank ein eigenes Couchapp-Makro entwickelt, was den Entwicklungsprozess beschleunigte.

5.3.3.2 Firefox / Firebug

In Abschnitt 8.5.2 wird erläutert, warum der Browser Firefox [Moz10d] in mindestens Version 3.5 als Zielplattform gewählt wurde. Aus diesem Grund wurde die Anwendung mithilfe der Firefox-Erweiterung Firebug [Moz10a] entwickelt.

Mit der Firebug können Stylesheets, HTML, das DOM und JavaScript auf einer Webseite untersucht werden. Eine Konsole erlaubt das Loggen von Log-Statements im JavaScript-Code und von HTTP-Requests. Der Quelltext einer Webseite kann ebenfalls live analysiert und auch editiert werden. Dies ermöglicht einfaches Debugging. Es gilt deshalb als besonders beliebt unter Webentwicklern [IDG10]. Firebug ist das am sechsthäufigsten heruntergeladene Add-on für Firefox [Moza]. Es wird von mehr als zwei Millionen Menschen täglich benutzt [Mozb].

Firebug wurde in diesem Projekt sowohl für Entwicklung des DOMs, für die Gestaltung des Frontends, als auch für einfache Performance-Optimierung des Seitenaufbaus verwendet.

6 Anforderungen an das System

Im Folgenden werden die Anforderungen an das System definiert. Funktionale und nichtfunktionale Anforderungen werden in schriftlicher Form festgelegt, eine tabellarische Übersicht findet sich im Anhang (Abschnitt A.4.1). Die erforderlichen und erwünschten Funktionalitäten werden dabei auch durch Use-Case-Diagramme beschrieben. In diesem Kapitel soll die Frage beantwortet werden, was ein System leisten muss, damit es die in der Analyse (Kapitel 3) beschriebenen Vorgaben erfüllt.

6.1 Funktionale Anforderungen

6.1.1 Muss-Kriterien

Abbildung 6.1 zeigt ein Use-Case-Diagramm für die in den Muss-Kriterien definierten Anforderungen an die Outline-Verwaltung und den Gliederungseditor. In Abbildung 6.2 finden sich die Anforderungen an Replikation und Konfliktbehandlung. Der im Rahmen dieser Arbeit entwickelte Prototyp soll folgenden Anforderungen in jedem Fall entsprechen. Nur dann kann von einer erfolgreichen Umsetzung der Aufgabenstellung gesprochen werden.

6.1.1.1 Outline-Verwaltung

Der Benutzer muss eine beliebige Anzahl von Outlines erstellen können (**FA100**). Die Outlines müssen übersichtlich dargestellt werden und einen veränderbaren Titel haben (**FA101**).

6.1.1.2 Gliederungseditor

Der Gliederungseditor muss das Look & Feel eines Texteditors mit einer beliebigen Anzahl von Zeilen haben (**FA200**). Zwischen den Zeilen muss mit den üblichen Tasten/Tasten-Kombinationen navigiert werden können (**FA201**). Es muss möglich sein, den Inhalt der Zeilen zu bearbeiten (**FA202**). Beim Verlassen einer Zeile wird diese automatisch gespeichert (**FA203**). Wird das Fenster geschlossen, während eine Zeile editiert wird, muss diese automatisch gespeichert werden. Alternativ kann der Benutzer auch vor dem Schließen des Fensters darauf hingewiesen werden, dass Datenverlust zu befürchten ist (**FA206**).

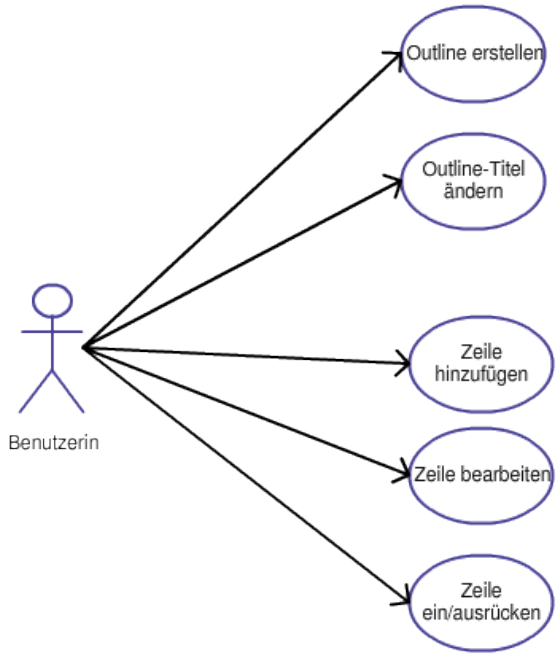

Abbildung 6.1: Use-Case-Diagramm für die Muss-Kriterien, Outline-Verwaltung

Die Zeilen müssen ein- und wieder ausrückbar sein, um eine hierarchische Abbildung zu ermöglichen (**FA204**). Das Ein- bzw. Ausrücken einer Zeile soll die unmittelbar darunter liegenden Zeilen mit tieferer Einrückung mitbewegen (**FA205**).

6.1.1.3 Replikation

Ist der Benutzer online oder wird nach einer Offline-Phase die Verbindung wiederhergestellt, müssen von ihm erstellte Outlines (**FA300**) und von ihm gemachte Änderungen an Outlines (**FA301**) sofort zum Server repliziert werden.

Der Benutzer muss alle im System von anderen Benutzern erstellten Outlines (**FA302**) und deren Änderungen an Outlines (**FA303**) automatisch auf seinen Rechner repliziert bekommen, sofern oder sobald diese mit dem Server verbunden sind. Er muss über Änderungen benachrichtigt werden, sobald diese vorliegen (**FA304**). Dabei soll der Arbeitsfluss nicht unterbrochen werden.

Wird nach einer Offline-Phase die Verbindung wiederhergestellt, muss der Benutzer entweder darauf hingewiesen werden, dass Replikation jetzt wieder möglich ist, oder die Replikation muss automatisch gestartet werden (**FA305**).

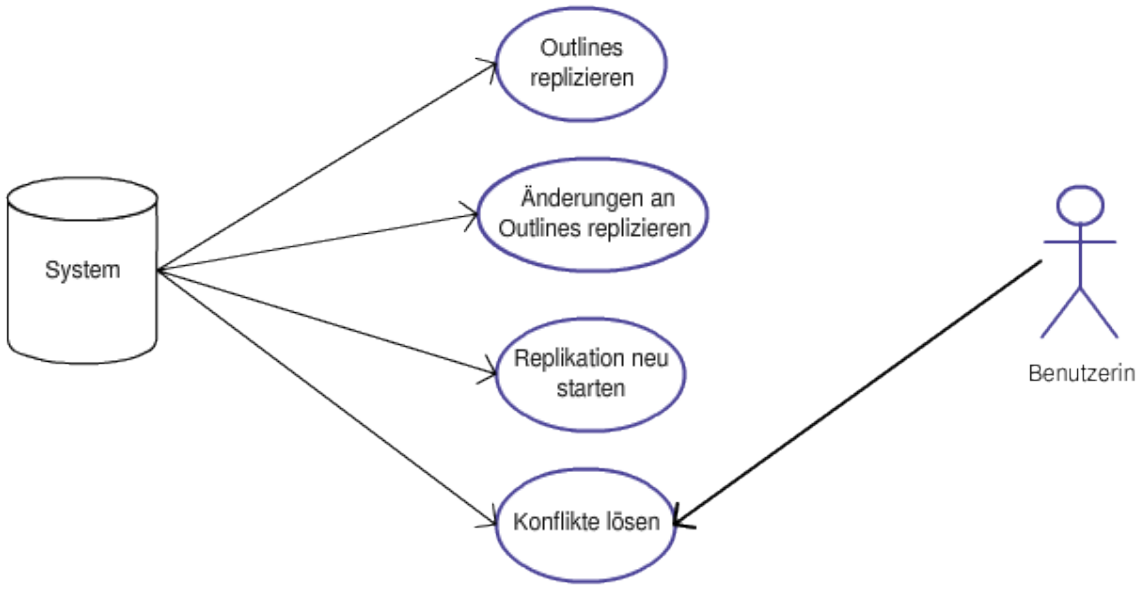

Abbildung 6.2: Use-Case-Diagramm für die Muss-Kriterien, Replikation

6.1.1.4 Konfliktbehandlung

Von den Konflikten, die beim Replizieren entstehen können, soll mindestens eine Art vom System selbstständig gelöst werden (**FA400**). Mindestens eine Konfliktart soll der Benutzer manuell lösen können (**FA401**).

6.1.2 Kann-Kriterien

Die in diesem Abschnitt aufgestellten Kriterien müssen nicht alle im Prototyp implementiert werden. Beim Design des Systems soll allerdings ihre spätere Umsetzbarkeit miteinbezogen werden.

Abbildung 6.3 zeigt ein Use-Case-Diagramm für die in den Kann-Kriterien definierten Anforderungen an die Outline-Verwaltung und den Gliederungseditor. In Abbildung 6.4 finden sich die Anforderungen an Replikation und Konfliktbehandlung.

6.1.2.1 Outline-Verwaltung

Outlines sollen gelöscht werden können (**FA102**).

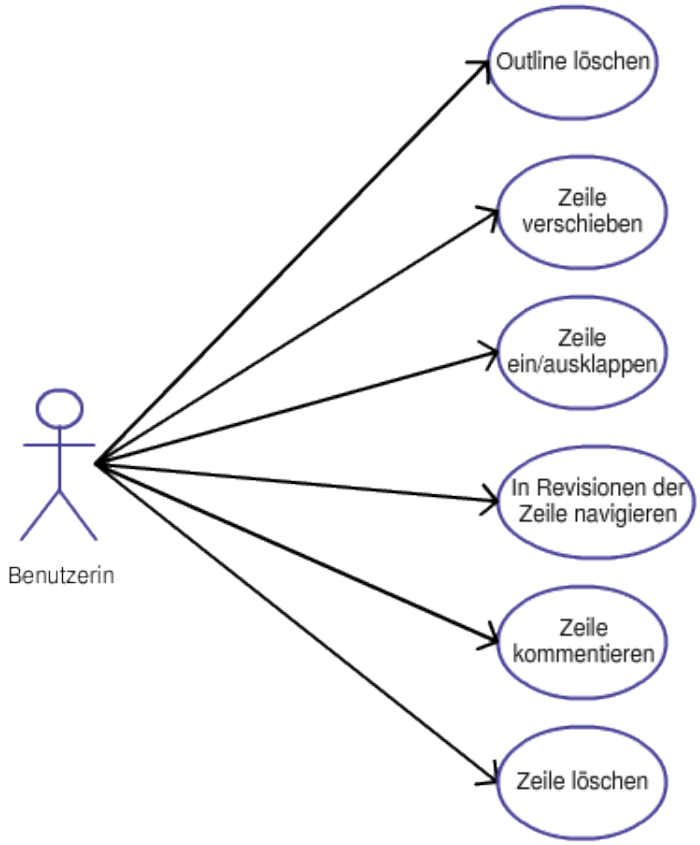

Abbildung 6.3: Use-Case-Diagramm für die Kann-Kriterien, Outline-Verwaltung

6.1.2.2 Gliederungseditor

Die Zeilen sollen nach unten und oben verschoben werden können (**FA207**). Die Größe der Zeile soll sich automatisch an die Menge des Textes anpassen (**FA208**). Für eine verbesserte Übersichtlichkeit sollen die Zeilen ein- und ausklappbar sein (**FA209**). Die Information, welche Zeilen eingeklappt sind, soll nicht mitrepliziert, möglichst aber lokal gespeichert werden (**FA210**).

Die Revisionen einer Zeile sollen automatisch gespeichert werden (**FA211**). Zwischen den Revisionen soll gewechselt werden können (**FA212**). Es soll möglich sein, die einzelnen Zeilen mit Kommentaren zu versehen (**FA213**) und die Zeilen löschen (**FA214**).

6.1.2.3 Replikation

Die zur Verfügung stehenden Outlines sollen vom System veröffentlicht werden, so dass mit dem Server verbundene Benutzer auswählen können, welche einzelnen Outlines sie replizieren möchten (**FA306**).

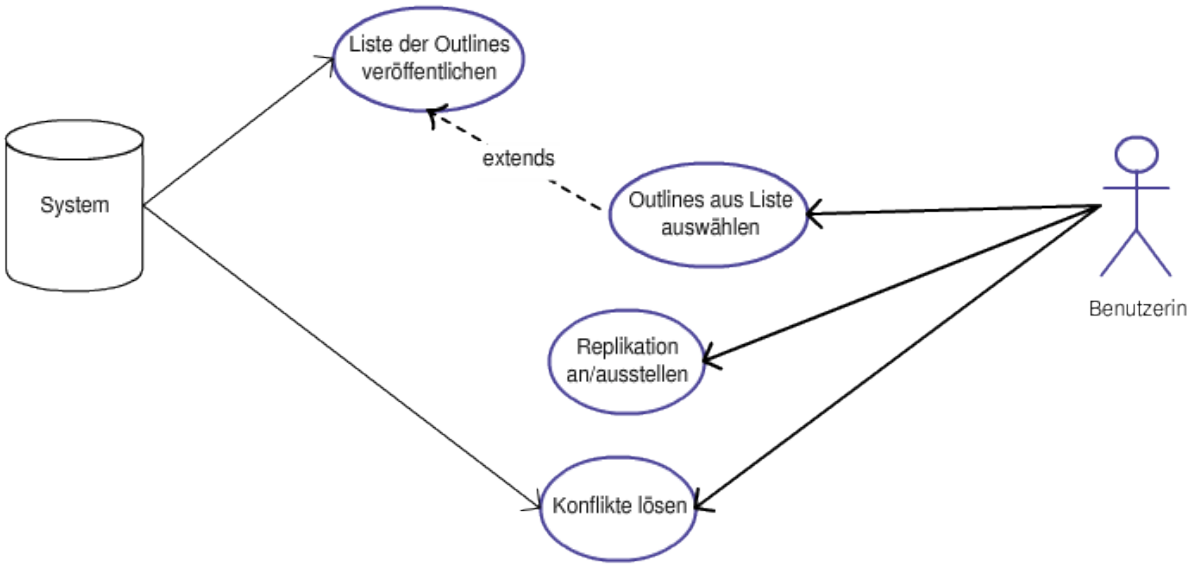

Abbildung 6.4: Use-Case-Diagramm für die Kann-Kriterien, Replikation

Der Benutzer soll über eine Statusmeldung informiert werden, ob gerade eine Verbindung zum Server besteht oder nicht (**FA307**). Über das Interface soll es die Möglichkeit geben, Replikation an- und auszustellen (**FA308**).

6.1.2.4 Konfliktbehandlung

Kombinationen aus unterschiedlichen Konfliktarten sollen vom System / vom Benutzer gelöst werden können (**FA402**). Konflikte, die zwischen mehr als zwei Repliken auftreten, sollen korrekt behandelt werden (**FA403**).

Es sollen möglichst viele Konflikte vom System selbstständig gelöst werden können (**FA404**). Die Oberfläche soll so entworfen sein, dass der Benutzer möglichst viele Konfliktarten manuell lösen kann (**FA405**).

6.1.3 Abgrenzungs-Kriterien

6.1.3.1 Outline-Verwaltung

Es wird keine Benutzerverwaltung und keine Zugriffsverwaltung für Outlines implementiert.

6.1.3.2 Gliederungseditor

Es werden keine Spalten implementiert.

6.1.3.3 Replikation

Es wird keine Peer-to-Peer-Replikation geben.

6.1.3.4 Konfliktbehandlung

Das System wird nur für die Benutzung durch eine kleine Anzahl Benutzer optimiert sein. Bei Konflikten zwischen mehr als zwei kollidierenden Versionen ist eine verlässliche Konfliktbehandlung nicht gesichert.

6.2 Nichtfunktionale Anforderungen

6.2.1 Einsatz

6.2.1.1 Zielgruppe

Benutzer des Systems müssen durchschnittliche Kenntnisse in der Bedienung eines Computers, insbesondere eines Webbrowsers haben. Des Weiteren müssen sie in der Lage sein, eine CouchDB-Instanz auf ihrem Rechner zu installieren und zu starten. Bei den Benutzern sollte ein Verständnis für die Vorteile und Grenzen der Einsatzmöglichkeiten des Systems vorhanden sein.

6.2.1.2 Betriebsbedingungen

Der Server für den Austausch der Outlines und der Updates muss stabil 24 Stunden am Tag und sieben Tage die Woche laufen, damit die Dienste jederzeit verfügbar sind. Um dies sicherzustellen, soll das Deployment mit dem Service Amazon Elastic Compute Cloud (Amazon EC2) vorgenommen werden. Für bessere Skalierbarkeit soll darüber hinaus das Clustering Framework CouchDB-Lounge eingesetzt werden.

Die Anwendung soll auf jedem Rechner bereitgestellt werden können, auf dem CouchDB installiert werden kann. Nähere Hinweise zu von CouchDB unterstützten Systemen finden sich in Abschnitt 9.1.

6.2.2 Umgebung

Um Lizenzgebühren zu vermeiden und das System an wechselnde Anforderungen anpassen zu können, soll es ausschließlich mit Open-Source Software umgesetzt werden.

6.2.2.1 Hardware

Wenn die Couch-Instanz, die als Server dienen soll, nicht mit Amazon EC2, sondern auf einem eigenen Server bereitgestellt wird, muss dieser folgenden Mindestanforderungen genügen:

- Intel-Prozessor mit Frequenz 3,0 Ghz
- mind. 5 GB Festplatte
- 1 GB Hauptspeicher
- LAN-Anschluss Fast Ethernet 100 MBit

6.2.2.2 Software

Das System baut auf mehreren Softwarepaketen und Programmiersprachen auf. Die angegebenen Versionen sind Mindestanforderungen. Hinweise zur Installation finden sich in Abschnitt 9.1.

- CouchDB 0.11.0
- Spidermonkey 1.7
- Erlang 5.6.5
- ICU 3.0
- cURL 7.18.0
- Automake 1.6.3
- Autoconf 2.59

Wenn die mitgelieferten Rake-Tasks für Deployment und Betrieb genutzt werden sollen, muss Ruby in der Version 1.8.6. oder größer installiert werden (s. Abschnitt 9.2.4).

Hinweise für das Testsetup finden sich in Abschnitt 8.8.

6.2.3 Benutzeroberfläche

Für das System soll eine einfache und übersichtliche Web-Oberfläche gestaltet werden. Die Aktivierung von JavaScript kann vorausgesetzt werden. Die Oberfläche soll mindestens im Browser Firefox in der Mindestversion 3.5 vollständig funktionieren. Die Verzögerung zwischen Eingabe auf der Weboberfläche und Eintreten der gewünschten Funktionen sollte möglichst weniger als eine Sekunde, in keinem Fall aber mehr als vier Sekunden betragen. Diese Zeitspanne wird in einer Studie als noch tolerierbare Wartezeit auf Antwort in zwischenmenschlichen Gesprächen bzw. in Telefonsystemen bezeichnet [Mil68, S. 267 bzw. 270]. Jakob Nielsen überträgt diese Zeitspanne auf Interaktionszeiten mit Webapplikationen [Nie93, Kap. 5.5].

Abbildung 6.5 zeigt ein Mockup für die Struktur der Seite mit der Outline-Übersicht. Abbildung 6.6 zeigt den Aufbau der Seite, die den Gliederungseditor enthält.

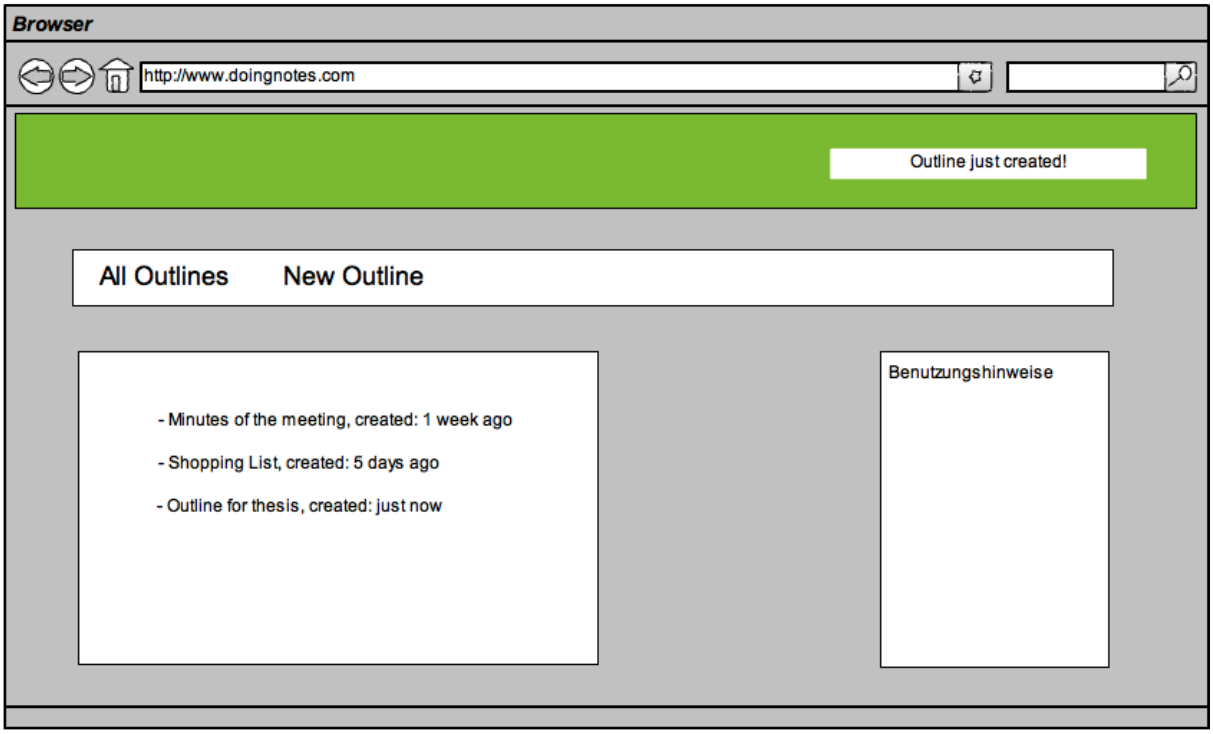

Abbildung 6.5: Struktur der Web-Oberfläche: Outline-Übersicht

6.2.4 Qualitätsziele

Die Qualität des erstellten Systems ist ein weiteres wichtiges Ziel. Die qualitativen Anforderungen sind folgende:

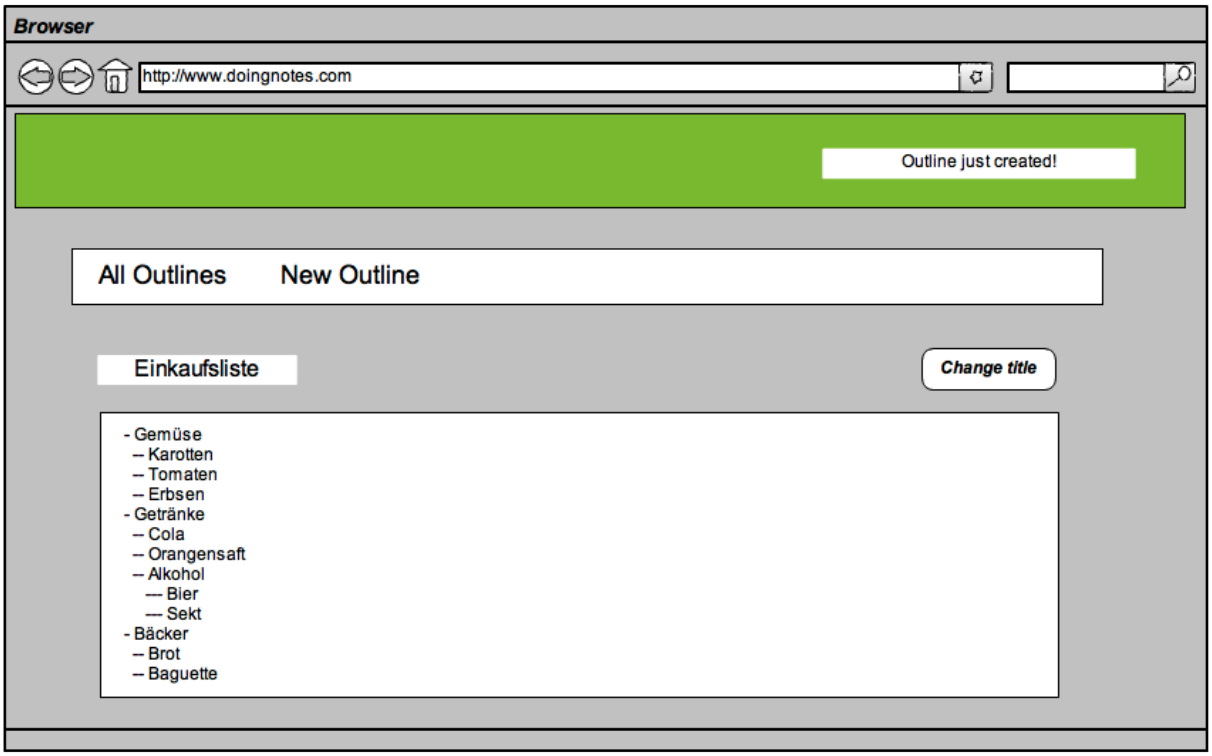

Abbildung 6.6: Struktur der Web-Oberfläche: Outline-Einzelansicht

- Erweiterbarkeit durch offene Architektur
- Portabilität durch geringe Hardware- und Software-Anforderungen
- Hohe Softwarequalität durch möglichst flächendeckende Testabdeckung
- Design und Programmierung orientiert an sprach- und frameworkspezifischen Standards
- Implementierung nach der MVC-Architektur: Trennung zwischen Oberfläche / Anwendungslogik / Datenhaltung
- Möglichst wartbarer, deshalb einfacher und redundanzfreier Code
- Einhaltung von Usability- und Accessability-Guidelines bei der Oberfläche
- Geringe Ladezeiten der Anwendung im Browser, deshalb möglichst wenig Abhängigkeit von externen Frameworks

7 Systemarchitektur

Aufbauend auf den Systemanforderungen in Kapitel 6 wird in diesem Kapitel der Entwurf der Systemarchitektur beschrieben. Da eine als Couchapp umgesetzte Anwendung eine Architektur ohne Middleware ermöglicht, liegt der Schwerpunkt weniger auf der Darstellung der Komponenten. Stattdessen wird die innere Struktur der Anwendung vorgestellt. Die in Frage kommenden Konzepte für die zentralen Problemstellungen bei der Konzeption werden diskutiert und die jeweils getroffene Wahl begründet. So wird ein Überblick über die Funktionsweise der Datenhaltung, der Anwendungslogik sowie der Benutzeroberfläche vermittelt.

Die Code-Beispiele in diesem Kapitel sind nicht die kompletten CouchDB-Dokumente. Sie enthalten lediglich die Teile, die für die Demonstration der jeweiligen Aspekte wichtig sind.

7.1 Gesamtüberblick über die Architektur

Klassische Webanwendungen sind nach der *Client-Server*-Architektur aufgebaut: Die Daten liegen in einer meist relationalen Datenbank, die Anwendungslogik wird auf dem Server ausgeführt und die Ergebnisse werden an den Client ausgeliefert (s. Abb. 7.1). Lediglich kleine Teile der Darstellungslogik werden manchmal als *Add-On* im Webbrowser abgearbeitet, um die Anwenderakzeptanz zu verbessern.

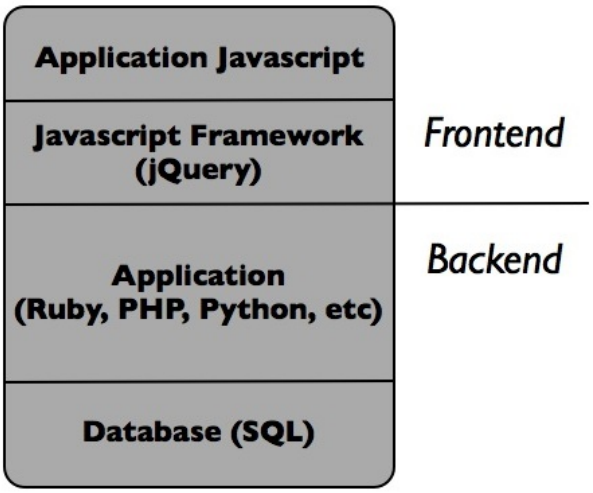

Abbildung 7.1: Architektur einer klassischen Webanwendung, nach [Qui09]

Setzt man nun voraus, dass der Browser JavaScript und HTML5 unterstützt, können auch größere Teile der Applikation lokal auf dem Rechner des Benutzers ausgeführt werden. CouchDB bringt darüber hinaus einen eigenen Webserver mit. Dadurch entfällt die Notwendigkeit, für die Applikationslogik eine Middleware zu erstellen (s. Abb. 7.2).

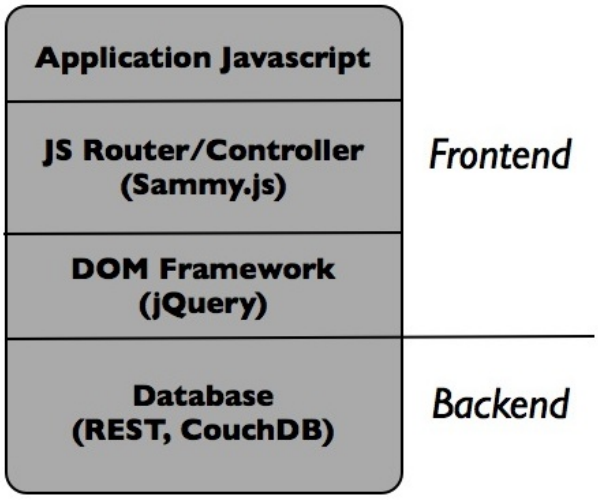

Abbildung 7.2: Architektur einer Couchapp, nach [Qui09]

Ist CouchDB auf dem lokalen Rechner installiert, kann die Anwendung wie ein Desktop-Programm benutzt werden. Die Instanz auf dem Server dient lediglich dazu, die Outlines zwischen den Clients zu synchronisieren (s. Abb. 7.3). Es reicht aus, eine einzige Anwendung zu implementieren, die auf den Clients sowie auf dem Server eingesetzt wird. Auf den Clients kann die Anwendung auch eingesetzt werden, wenn der Server nicht erreichbar ist.

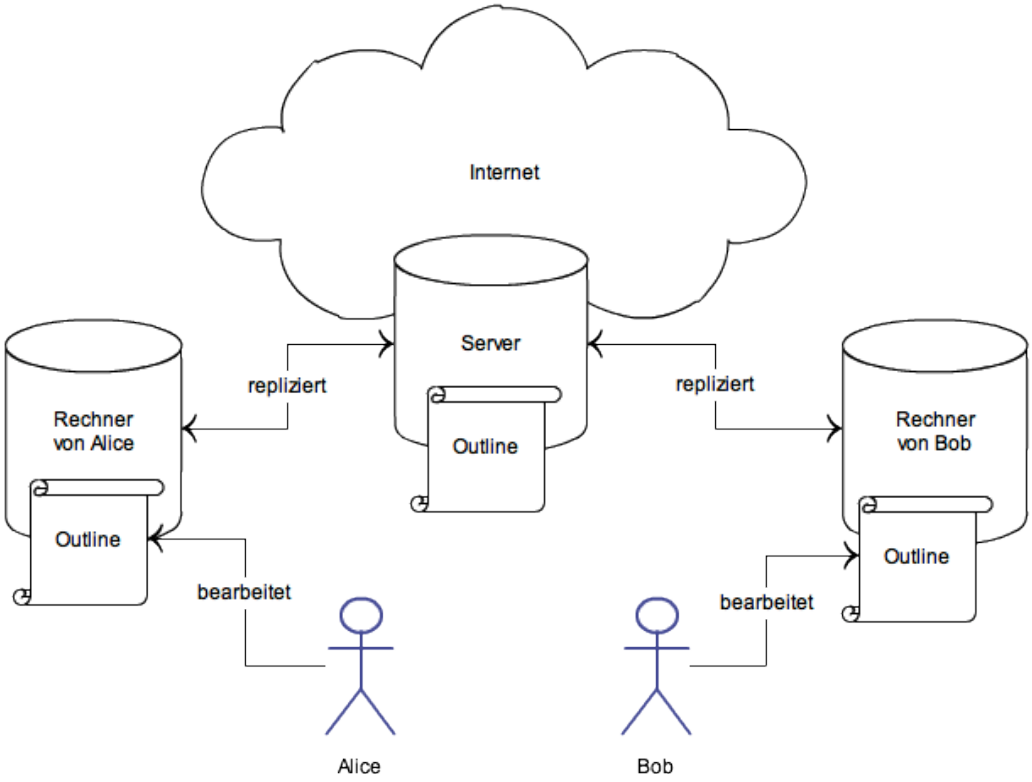

Abbildung 7.3: Projektvision

7.2 Modellierung der Datenstruktur

Für die Abbildung der Daten in der Datenbank kamen mehrere Alternativen in Frage. Durch die Diskussion dieser Möglichkeiten wird die gewählte Lösung begründet.

7.2.1 Anforderungen

Die Anforderungen an den Entwurf der Datenmodellierung werden in folgender Priorität definiert:

- Die Programmierung und Wartung der Anwendung soll möglichst einfach sein.
- Konflikte beim gleichzeitigen Speichern von Daten sollen möglichst vermieden werden oder nur selten auftreten.
- Der Zugriff auf die Daten soll möglichst performant sein. Da weniger Abfragen zu kürzeren Zugriffszeiten führen, sollen die zusammengehörenden Daten möglichst auch zusammen gespeichert werden.

CouchDB ermöglicht eine Replikation, bei der konflikthafte Dokumente automatisch markiert werden. Für eine benutzbare Anwendung muss jedoch eine Anwendungslogik umgesetzt werden, die die Auflösung dieser Konflikte ermöglicht. Für den Entwurf der Datenstruktur wurden Hinweise aus [Cou09] entnommen.

7.2.2 Problemstellung

Ein Outline ist eine sortierte, hierarchisch verschachtelte Liste von unterschiedlich tief eingerückten Zeilen. Ein einfaches Beispiel findet sich in Abbildung 7.4. Dabei sind die Aufzählungszeichen für Zeilen ohne Kindknoten Kreise, für Zeilen mit Kindknoten Dreiecke. Ein solches Outline soll möglichst entsprechend der oben genannten Anforderungen umgesetzt werden.

Im Beispiel wird mithilfe des Outliners eine Einkaufsliste umgesetzt. Dies stellt sicher nicht den zentralen Anwendungsfall für einen Gliederungseditor dar; dafür wird die hierarchische Einrückung intuitiv deutlich, und das Beispiel kann kurz, aber realistisch gehalten werden.

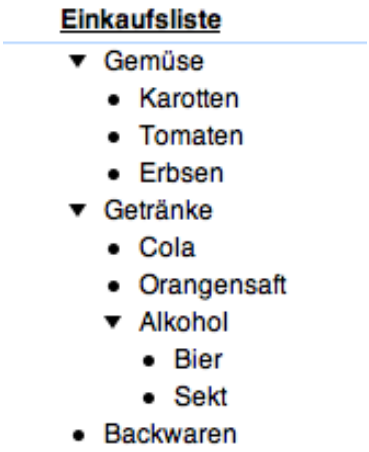

Abbildung 7.4: Einfaches Outline

7.2.3 Speicherung in einem JSON-Dokument

Die einfachste Implementierung ist die Umsetzung in einem einzigen JSON-Dokument (s. Listing 7.1). Bei diesem Entwurf müssen die Zeilen wiederum eigene JSON-Objekte sein, damit sie geschachtelt gespeichert werden können.

Von diesem Outline werden während der Lebensdauer der Anwendung zeitweise zwei (oder mehr) Versionen existieren, die sich voneinander unterscheiden. Bei der nächsten Synchronisierung der beiden Versionen sollen diese wieder zusammengeführt werden. Zeilen, die verändert, hinzugefügt

oder gelöscht werden, sollen auch in der anderen Version verändert, hinzugefügt oder gelöscht werden. Der Benutzer muss nur intervenieren, wenn für dieselbe Zeile zwei konkurrierende Versionen vorliegen. Wie kann dies erreicht werden?

```
{
  "title": "Einkaufsliste",
  "lines": [
    {"text": "Gemüse", "lines": [
      {"text": "Karotten"},
      {"text": "Tomaten"},
      {"text": "Erbsen"}
    ]},
    {"text": "Getränke", "lines": [
      {"text": "Cola"},
      {"text": "Orangensaft"},
      {"text": "Alkohol", "lines": [
        {"text": "Bier"},
        {"text": "Sekt"}
      ]},
    ]},
    {"text": "Backwaren"}
  ]
}
```

Listing 7.1: Einfaches Outline in einem JSON-Dokument

Die Struktur in Listing 7.1 ist bereits valides JSON. In CouchDB könnte es in dieser Form in einem einzigen Dokument gespeichert werden. Dieses Dokument könnte dann mit einem einzigen Lesezugriff gelesen werden. Für die Replikation wird ebenfalls nur ein einziger Vorgang benötigt. Auch die Umsetzung der Anwendungslogik ist vergleichsweise simpel, da die für das Sortieren und das Einrücken der Zeilen nötigen Informationen schon im Dokument enthalten sind.

Probleme treten jedoch auf, sobald das Dokument modifiziert wird. Wenn eine Zeile verändert, hinzugefügt oder verschoben wird, speichert CouchDB für das Outline-Dokument eine neue Version mit einer neuen Revisionsnummer. Ein Konflikt entsteht ausnahmslos jedesmal, wenn das Outline nach der Modifikation mit der Version eines anderen Benutzers repliziert wird.

Auch nach einer Replikation mit vielen Änderungen im Dokument gibt es nur zwei verschiedene Sets von Zeilen. Ein Set wird als die Gewinner-Revision, das andere als die konflikthafte Revision gespeichert werden. Diese Semantik ist sehr unpraktisch für den Benutzer: Er kann nur noch seine eigene oder die andere Version zum Behalten auswählen. So müssen sämtliche Änderungen wieder manuell angewendet werden. Zentrale Vorteile der CouchDB-Replikation sind somit verfallen.

7.2.4 System Prevalence

Eine weitere Möglichkeit der Umsetzung ist die Anwendung von *System Prevalence* [Pat05]. Diese Persistierungstechnik wird in Objektdatenbanken wie Madeleine [Ben06] angewandt. Anstatt neuer Versionen des Outlines werden Operationen auf einem Ausgangsdatensatz gespeichert. So sind „Ändern", „Speichern", „Verschieben" einer Zeile einzelne Einträge in der Geschichte des Outlines. Diese Einträge werden nacheinander gespeichert und nie verändert. So entstehen niemals Konflikte bei der Replikation. Die Einträge können in einem Array in einem einzelnen CouchDB-Dokument gespeichert werden. Wenn bei der Replikation Konflikte auftreten, sind diese leicht zu lösen, indem die Elemente der beiden Arrays kombiniert werden.

Probleme treten auch hier auf, denn wenn unterschiedliche Versionen zusammengeführt werden, hat die Reihenfolge der Änderungen Auswirkungen auf das Ergebnis. Es müsste ein Algorithmus entwickelt werden, der die Reihenfolge der Änderungen sinnvoll festlegt.

7.2.5 Speicherung der Versionshistorie

Anstatt die Historie der Kommandos zu speichern, könnten auch die alten Revisionen der Outlines gespeichert werden. CouchDB löscht alte Revisionen nach einem `compact`-Vorgang der Datenbank. Um dies zu verhindern, könnten die Revisionen in eigene Dokumente persistiert werden. Das System muss eine Referenz auf die aktuellste Revision speichern, jede Revision muss auf ihren Vorgänger verweisen. Beim Zusammenführen nach einem Replikationsvorgang könnten die abweichenden Versionen so lange verglichen werden, bis der letzte gemeinsame Vorgänger gefunden ist. Auf dieser Basis kann dann das Zusammenführen stattfinden.

Dieses Verfahren wird von Versionskontrollsystemen wie Git oder Subversion angewendet. Hier müssten entsprechend JSON-Felder anstelle von Zeilen innerhalb von Textdateien verglichen werden.

Mit dieser Vorgehensweise wird also für jeden Schreibvorgang eine komplette Version des gesamten Outlines gespeichert. Dabei steigt der Umfang der Daten in der Datenbank stark an. Dies ist ein beträchtlicher Nachteil. Beim Zusammenführen müssen anwendungsseitig beide Versionen der Dokumente zerlegt werden, um feststellen zu können, in welcher Version welche Zeile auf welche Weise verändert wurde. Dieses Vorgehen wurde mithilfe eines einfachen Prototypen mit geringem Funktionsumfang getestet, jedoch schnell als zu komplex verworfen. Stattdessen sollte ein Verfahren entwickelt werden, bei dem die Zerlegung schon im Dokument vorgenommen wird, um das Qualitätsziel der möglichst geringen Komplexität zu beachten.

7.2.6 Ausgliedern der Zeilen in einzelne JSON-Dokumente

Die letzte untersuchte Möglichkeit ist, jede Zeile in einem eigenen Dokument zu speichern. Das Hinzufügen oder Löschen einer Zeile wird über das Erstellen oder Löschen eines JSON-Dokumentes umgesetzt. Aus diesem Vorgang entsteht nicht per se ein Konflikt. Das Ändern einer Zeile wird nur dann zu einem Konflikt führen, wenn beide Seiten gleichzeitig dieselbe Zeile verändern. Erst diese Situation erfordert die Intervention eines Benutzers. Das Thema der Replikation ist so relativ trivial in der Anwendung zu lösen.

Aus dieser Art, das Problem zu modellieren, resultieren viele kleine Dokumente in der Datenbank, die jeweils nur eine Zeile enthalten. Deswegen muss ein Attribut eingeführt werden, mit dem Zeilen von Outlines in der Datenbank unterschieden werden können. Jedes Zeilen-Dokument enthält eine Referenz auf das Outline, zu dem es gehört.

Mit dieser Herangehensweise ergibt sich das Problem, dass die Sortierung und die Schachtelung der Zeilen nicht mehr innerhalb des Dokumentes persisiert sind. Die Lösung für dieses Problem wird in Abschnitt 7.3 eigens behandelt.

7.2.7 Fazit

Alle vier Lösungen haben gemeinsam, dass die Historie der Daten in irgendeiner Form gespeichert werden muss, denn nur so kann beim Zusammenführen der Repliken der aktuelle Zustand der Daten mit dem Zustand verglichen werden, in dem sich die Versionen noch nicht unterschieden haben. CouchDB speichert verschiedene Versionen von Dokumenten; es ist sinnvoll, sich dieses Feature zunutze zu machen. Der einfachste und am wenigsten fehleranfällige Weg ist daher, die Zeilen als eigene Dokumente zu speichern. Dies wird in Listings 7.2 und 7.3 beispielhaft dargestellt.

```
{
  "_id": "1dbdcbc27b22cc7a14cd48d397000657",
  "kind": "Outline",
  "title": "Einkaufsliste"
}
```

Listing 7.2: Outline mit ID und Typ

```
{
   "kind": "Line",
   "text": "Gemüse",
   "outline_id": "1dbdcbc27b22cc7a14cd48d397000657"
},
{
   "kind": "Line",
   "text": "Getränke",
```

```
    "outline_id": "1dbdcbc27b22cc7a14cd48d397000657"
},
{
    "kind": "Line",
    "text": "Backwaren",
    "outline_id": "1dbdcbc27b22cc7a14cd48d397000657"
}
```

Listing 7.3: Drei Zeilen mit ID und Typ

Durch die Erstellung einer CouchDB-View mit einem zusammengesetzten Schlüssel können Outline und alle ihre Zeilen in nur einem Request abgefragt werden. Für dieses Problem wird die weit verbreitete Technik der *View-Collation* eingesetzt, die das Bilden von Joins simuliert [Leno7]. Mehr zum Einsatz von Views siehe 4.2.6. Der Schlüssel dieser View ist ein JSON-Array, dass sich aus der Outline-ID und einer Zahl zusammensetzt. Die Zahl ist bei Dokumenten mit Typ `Outline` eine 0, bei Dokumenten mit Typ `Line` eine 1. Da die Schlüssel die Kollation (für die Sortierreihenfolge) der Zeilen beeinflussen, wird das erste Element des resultierenden Arrays immer das Outline sein. Erst darauf folgen alle Zeilen. Das Dokument ist der Value eines jeden Array-Elements. Mit diesem definierten Ergebnis kann nun die Anwendung die Ausgabe eines Outlines mit allen zugehörigen Zeilen vornehmen.

Die View ist in Listing 7.4 angegeben. Abgefragt wird sie mit der Outline-ID als Key-Parameter. Dadurch werden nur dieses Outline und die zugehörigen Zeilen ausgegeben: `http://localhost:5984/doingnotes/_design/doingnotes/_view/notes_by_outline?key="01234567890"`. Im diesem Beispiel hat das Outline die ID „01234567890“.

```
function(doc) {
  if (doc.kind == "Outline") {
    emit([doc._id, 0], doc);
  } else if (doc.kind == "Line") {
    emit([doc.outline_id, 1], doc);
  }
}
```

Listing 7.4: View zum Ausgeben aller Zeilen zu einem Outline

Nachdem die Entscheidung für den Ansatz der Datenmodellierung getroffen wurde, bleibt das Problem zu lösen, wie Sortierung und Einrückung der Zeilen am besten abzubilden sind. Dies wird im nächsten Abschnitt diskutiert.

7.3 Umsetzung der Zeilensortierung und -einrückung

Es muss ein im Bezug auf die in Abschnitt 7.2.1 genannten Anforderungen effektiver Weg gefunden werden, wie Reihenfolge und Einrückungsgrad der Zeilen in einem Outline gespeichert werden können. Das Ziel ist die Abbildung der in Abbildung 7.4 beschriebenen Struktur. Mehrere Möglichkeiten, dieses Problem zu lösen, sind denkbar. Es ist zu unterscheiden, ob es vorteilhafter ist, die Zeilen zu verbinden, sei es in Form einer verketteten Liste oder als Baumstruktur, oder ob es praktikabler ist, die Sortierreihenfolge in den Zeilen festzulegen. In diesem Abschnitt werden die Vor- und Nachteile für beide Ansätze abgewogen.

Die Anforderungen an die Lösung sind, dass das Einfügen, Löschen und Verschieben einer Zeile mit möglichst wenig Schreibzugriffen verbunden ist. Außerdem soll die Position einer Zeile immer genau definiert sein; Zeilen dürfen weder denselben Platz beanspruchen, noch darf die Information über ihre Positionierung verlorengehen.

7.3.1 Indiziertes Array

Der erste untersuchte Ansatz ist, die Zeilen als ungeordnete Liste zu speichern und ihnen einen Index zuzuweisen. Die Information über den Grad der Einrückung muss bei dieser Vorgehensweise ebenfalls explizit gespeichert werden.

7.3.1.1 Sortierung

Listing 7.5 ist ein Beispiel für den Einsatz von Sort-IDs.

```
{
  "text": "Gemüse",
  "sort_id": "1"
},
{
  "text": "Getränke",
  "sort_id": "2"
},
{
  "text": "Backwaren",
  "sort_id": "3"
}
```

Listing 7.5: Drei Zeilen mit einfachem Index

Wenn nun ein Element eingefügt wird, erhält das Element die `Sort-ID` des nachfolgenden Elements, und jedem der nachfolgenden Elemente muss ein neuer Index zugewiesen werden. Dies

ist problematisch, weil die Anzahl der Schreibzugriffe bei zunehmender Länge des Dokuments sehr schnell ansteigt. Die trivialste Lösung wäre, Indizes mit sehr hohem Abstand zu vergeben (0, 1000, 2000). Dies ist jedoch kein skalierbarer Ansatz: Im ungünstigsten Fall müssen Indizes sehr schnell mehrfach zugewiesen werden, wenn mehrere Elemente an derselben Stelle eingefügt werden.

Eine andere Möglichkeit ist es, für den Index den Datentyp *Floating Point* zu wählen. Dieser Ansatz wird in [And10, Kap. 24] vorgeschlagen. Beim Einfügen erhält ein Element als Index den Mittelwert zwischen den Indizes der beiden umgebenden Elemente. Am Beispiel von Listing 7.6: Wird eine Zeile zwischen `Gemüse` und `Getränke` eingefügt, erhält sie als Index $\frac{(0.2+0.3)}{2} = \frac{0.5}{2} = 0.25$.

```
{
  "text": "Gemüse",
  "sort_id": 0.1
},
{
  "text": "Getränke",
  "sort_id": 0.2
},
{
  "text": "Backwaren",
  "sort_id": 0.3
}
```

Listing 7.6: Drei Zeilen mit Float-Index

Der Vorteil dieser Herangehensweise ist, dass für Verschieben und Einfügen einer Zeile nur ein einziger Schreibzugriff notwendig ist.

Das Problem an diesem Ansatz ist, dass die Präzision des Datentyps Float begrenzt ist. Bei häufigem Verschieben der Zeilen würde die maximale Anzahl der Nachkommastellen schnell erreicht werden. Indizes würden wieder mehrfach vergeben, wenn die Anzahl der Zeilen eines Outlines ansteigt. Dies könnte mit einem regelmäßigen Zurücksetzen der Indizes auf Zahlen mit geringerer Anzahl Stellen verhindert werden. Dies ist für ein verteiltes Setup jedoch nicht praktikabel, da bei dem Zurücksetzen alle anderen Benutzer einen Hinweis auf Änderungen im gesamten Outline erhalten würden. Darüber hinaus würden durch diese Operation zahlreiche Konflikte entstehen.

In [Cou09] wird empfohlen, den Index als String zu implementieren, dem bei jedem Verschieben ein Zeichen hinzugefügt wird. So umgeht man das Problem der Float-Präzision. Allerdings würde die Länge der Strings mit der Zeit ebenfalls stark ansteigen, was am Ende zu einem ähnlichen Problem wie bei der Umsetzung mit Floating Points führen würde.

Der Vorteil beider Herangehensweisen ist, dass für Verschieben und Einfügen einer Zeile nur ein einziger Schreibzugriff notwendig ist.

7.3.1.2 Einrückung

Sind die Zeilen nun in einer sortierten Liste gespeichert, müssen sie noch mit Information über den Grad der Einrückung versehen werden. Beim Ausgeben des Outlines wird diese Information ins DOM übertragen.

Dafür gibt es zwei Ansätze. Eine Zeile erhält entweder ein Attribut mit der Information, wie weit sie eingerückt ist (`"indent": 0`, `"indent": 1`, `"indent": 2`). Oder ihre Einrückungsdifferenz nach oben wird als Zahl gespeichert (`"indent": 0`, `"indent": 1`, `"indent": -1`). Der letztere Ansatz hat den Nachteil, dass beim Einrücken einer Zeile alle nachfolgenden Zeilen verändert werden müssen.

Der Nachteil beider Herangehensweisen ist, dass beim Ein- bzw. Ausrücken einer Zeile die unmittelbar darunter liegenden Zeilen mit tieferer bzw. gleicher Einrückung mitbewegt werden sollen. Für alle diese Zeilen muss dann ebenfalls ein eigener Schreibzugriff stattfinden. Je nach Beschaffenheit des Outlines kann die Anzahl der Schreibzugriffe stark ansteigen.

7.3.2 Verkettete Liste

Eine Alternative zur Umsetzung als indiziertes Array ist eine einfach verkettete Liste. Eine verkettete Liste ist eine Datenstruktur, in der die Objekte linear angeordnet sind. Im Gegensatz zu einem Array, bei dem die Reihenfolge von den Array-Indizes bestimmt wird, wird die Reihenfolge in einer verketteten Liste von Zeigern in jedem Objekt bestimmt [Coro1, Kap. 10.2].

Bei diesem Ansatz entfällt das Problem mit der (Neu-)Vergabe der Indizes beim Verschieben oder Einfügen von Elementen. Das Einfügen einer Zeile erfordert zwar einen Schreibzugriff zusätzlich zum Speichern der Zeile, da der neue Vorgänger seinen Zeiger auf die Zeile richten muss. Dies ist aber die maximale Anzahl der Schreibzugriffe, unabhängig von Komplexität und Länge des Outlines. Das Verschieben einer Zeile kann ebenfalls mit maximal zwei Schreibzugriffen umgesetzt werden.

Allerdings muss bei diesem Weg die Einrückung einer Zeile genauso umgesetzt werden wie bei der Umsetzung als indiziertes Array (s. Abschnitt 7.3.1.2). Die Nachteile bei den beschriebenen Lösungsansätzen hierfür bleiben bestehen.

7.3.3 Baumstruktur

7.3.3.1 Terminologie

Als Argument für die Modellierung als Baumstruktur sind die oben genannten Nachteile insbesondere bei der Einrückung einer sortierten Liste zu nennen. Baumstrukturen gehören zu den wichtigsten Datenstrukturen der Informatik. Laut [Knu97] handelt es sich um die wichtigsten nichtlinearen Strukturen unter den Computer-Algorithmen. Den Erfinder dieser Datenstruktur festzustellen ist nicht möglich [Gü04, S. 89]. Als Baumstruktur wird generell eine „verästelte" Beziehung zwischen Knoten bezeichnet.

Die Terminologie für Elemente von Bäumen ist in der Literatur nicht einheitlich. In [Knu97] werden folgende Bezeichnungen verwendet: Jede Wurzel ist *Elternknoten (parent)* von den Wurzeln der Teilbäume. Unmittelbar benachbarte Knoten sind untereinander *Geschwister (siblings)*, und *Kinder (children)* ihrer Eltern. Außerdem werden die Begriffe *Vorgänger (ancestor)* und *Nachkomme (descendant)* eingesetzt, um eine Beziehung zu beschreiben, die mehrere Ebenen eines Baums umspannt.

Laut der Klassizierung in [Knu97, Kap. 2.3.4] handelt es sich bei dem hier eingesetzten Baum um einen *Finite labeled rooted ordered Tree*. Jeder Knoten muss dabei einen Elternknoten haben; der Knoten ohne Elternknoten wird als Wurzel bezeichnet. Zyklische Beziehungen sind nicht erlaubt; vom Wurzelknoten aus ist jeder Knoten durch genau einen gerichteten Pfad erreichbar.

7.3.3.2 Umsetzung

In [Cor01, Kap. 10.4], werden Verfahren vorgestellt, wie eine Baumstruktur mithilfe von Zeigern umgesetzt werden kann. Das für das vorliegende Problem am besten geeignete Verfahren wird hier beschrieben. Es handelt sich um die *left child, right sibling*-Repräsentierung. Jeder Knoten ist dabei durch eine ID identifiziert. Jeder Knoten enthält einen Zeiger zu seinem ganz linken Kind, dem in einer vertikalen Abbildungsweise mit der Wurzel oben dem ersten Kind entspricht, und einen Zeiger zu seinem rechten Geschwisterknoten, also zu dem in der gleichen Ebene liegenden nächsten Knoten.

Der Vorteil an diesem Verfahren ist, dass die Anzahl der Schreibzugriffe bei jeder Art Operation zwei nicht übersteigt. Dies schließt das Ein- und Ausrücken eines ganzen Zeilenblocks mit ein.

7.3.4 Fazit

Nach Abwägen der Vor- und Nachteile wurde entschieden, die Anwendung mit einer Baumstruktur umzusetzen. Die konstante Anzahl der Schreibzugriffe beim Einsatz dieser Datenstruktur gab

den Ausschlag.

Mit einer View können ein Outline und alle zugehörigen Outlines auf einmal abgerufen werden, dies wurde bereits in Abschnitt 7.2.7 skizziert. Es ist ein für die Anwendung gewünschtes Verhalten, dass dies auf einmal geschieht. Die Zeilen werden nicht etwa erst bei Bedarf geladen, z. B. beim Ausklappen der Zeilen. Wie in Abschnitt 6.1.1.2 spezifiziert, soll der Outliner einem Texteditor ähneln, bei dem man auf den ersten Blick das gesamte Dokument erfassen kann und nur bei Bedarf einzelne Abschnitte verbirgt. Für die Ausgabe der Zeilen eines Outlines muss eine rekursive Funktion implementiert werden.

Mit einer leichten Abwandlung soll das Verfahren eingesetzt werden, das in Abschnitt 7.3.3.2 zuletzt beschrieben wurde. Jeder Knoten enthält statt eines Zeigers zu seinem ersten Kind einen Zeiger zu seinem Elternknoten. So muss beim Ein- oder Ausrücken einer Zeile nur der ein- oder ausgerückte Knoten verändert werden, der neue Elternknoten bleibt unverändert. Die Zeigerstruktur sieht demnach aus wie in Abbildung 7.5.

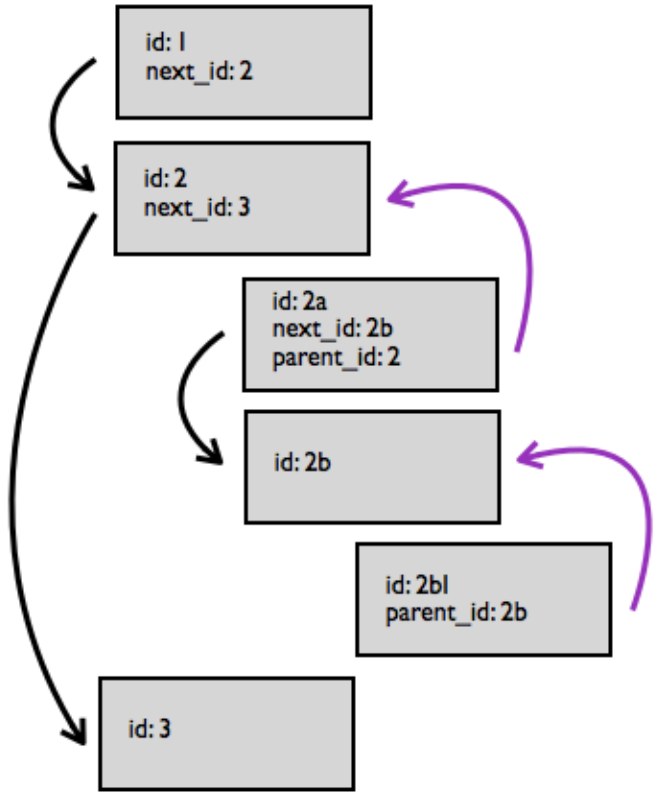

Abbildung 7.5: Die Zeiger-Struktur

Mit dem Umsetzen der Zeigerstruktur sieht das Beispiel-Outline aus Abbildung 7.4 am Ende aus wie in den Listings 7.7 und 7.8.

```
{
  "_id": "01234567890",
  "kind": "Outline",
  "title": "Einkaufsliste"
}
```

Listing 7.7: Gewählte Implementierung eines Outlines

```
{
   "_id": "111",
   "kind": "Line",
   "text": "Gemüse",
   "outline_id": "01234567890",
   "next_id": "333",
   "first_note": true
},
{
   "_id": "222",
   "kind": "Line",
   "text": "Karotten",
   "outline_id": "01234567890",
   "parent_id": "111"
},
{
   "_id": "222",
   "kind": "Line",
   "text": "Getränke",
   "outline_id": "01234567890"
}
```

Listing 7.8: Gewählte Implementierung von drei Zeilen

Damit ist die Frage nach der Modellierung der Datenstruktur beantwortet. Es bleibt festzulegen, wie mit den unvermeidlich auftretenden Konflikten umgegangen wird.

7.4 Konfliktbehandlung

Bereits 1996 entwarf Leslie Knieb ein verteiltes Datenbanksystem, das ein verteiltes Speichern von Daten ohne Locking-Mechanismus ermöglichte [Kli96]. Insbesondere ein Mergen der Daten und Lösen der Konflikte war mit diesem System ermöglicht. Auch wenn sich die Implementierung von der hier vorgestellten Technologie unterscheidet, die Anforderungen sind dieselben.

Knieb beschreibt drei Fälle von Datensynchronisation, in denen es *nicht* zu Konflikten kommt:

Änderung am Dokument, Neuanlegen eines Dokuments, und Löschen eines Dokuments [Kli96, Kap. 3].

Das Löschen wird in CouchDB über ein Update am Dokument implementiert. Das Dokument wird also nicht tatsächlich gelöscht, es erhält lediglich ein `deleted=true`-Attribut. Deshalb wird „Löschen“ hier nicht als Sonderfall betrachtet, sondern unter „Update“ mitbehandelt. Dies vereinfacht die Unterscheidung von Fällen, in denen Konflikte auftreten.

Im Folgenden werden die verbleibenden zu behandelnden Konflikte sowie der Umgang mit ihnen beschrieben. „Gleichzeitig“ meint im Folgenden den Zeitraum zwischen zwei Replikationsvorgängen von zwei oder mehr Repliken.

7.4.1 Gleichzeitiges Einfügen einer Zeile

Wird eine Zeile neu angelegt, ist diese Zeile immer konfliktfrei. Wenn mehrere Benutzer aber an der gleichen Stelle gleichzeitig eine neue Zeile anlegen, hat die vorhergehende Zeile (im Folgenden wie im Quelltext `previous` genannt) einen Konflikt, weil ihr *Next-Zeiger* sich ändert. Dieser Konflikt wird *Append-Konflikt* genannt. Es gibt jetzt zwei Versionen von `previous`, die auf jeweils eine der beiden neu eingefügten Zeilen zeigen.

Ein Append-Konflikt wird gelöst, indem das System die beiden neuen Zeilen, nach Datum absteigend sortiert, in das Outline einbaut. Entscheidend ist der Timestamp im Zeilen-Dokument. Die zeitliche Sortierung ist für viele Anwendungsfälle sinnvoll, und für die Benutzer nachvollziehbar. Da jedoch kein Verlass darauf ist, dass die Uhren auf den beiden Clients gleich gehen, ist dies in erster Linie ein Mechanismus, diesen Konflikt auf unterschiedlichen Clients eindeutig zu lösen. Auf diese Weise werden Situationen ausgeschlossen, in denen ein Konflikt auf zwei Clients gleichzeitig auf unterschiedliche Weise gelöst wird, was fälschlicherweise zu einer erneuten Konflikterkennung auf beiden Clients führen würde.

7.4.2 Gleichzeitiges Ändern einer Zeile

Wird der Text eines Dokumentes von mehr als einem Benutzer gleichzeitig geändert, entsteht ein Konflikt. Dies ist auch der Fall, wenn der neue Text in beiden Versionen gleich ist, da sich die neue Revisionsnummer in jedem Fall unterscheidet. Dieser Konflikt wird im Folgenden *Write-Konflikt* genannt.

Zur Auflösung wird dem Benutzer anstelle der konflikthaften Zeile ein Formular gezeigt, in dem beide Versionen des Dokuments zur Auswahl stehen. Er kann sich für eine Version entscheiden und diese auch noch editieren, also ggf. die beiden Versionen manuell zusammenfügen.

7.4.3 Weitere Konfliktarten

Als Sonderfall der Änderung an einem Dokument bleibt noch die Einrückung zu nennen. Wenn z. B. ein Benutzer eine Zeile einrückt, wird dies nach einer Replikation auch in den Dokumenten der anderen Benutzer sichtbar. Wenn aber der eine Benutzer eine Zeile ein- und die andere sie ausrückt, muss entschieden werden, wie dieser Konflikt den Benutzern präsentiert wird. Eine sinnvolle Standardlösung ist davon abhängig, wie der Gliederungseditor konkret benutzt wird.

Darüber hinaus sind Mischformen möglich, z. B. dass eine Zeile geändert und die nachfolgende Zeile eingerückt wird. Wenn bei der Editierung der Zeile ein Write-Konflikt auftritt, und sich beim Lösen für die alte Version entschieden wird, muss bei der Konfliktlösung auch der veränderte Next-Zeiger der Zeile berücksichtigt werden.

Konflikte, die durch Einrückung entstehen, werden bei der Entwicklung des Prototypen nicht berücksichtigt. Ebenfalls wird zur Vereinfachung davon ausgegangen, dass Konflikte nur zwischen zwei Versionen auftreten. Dies ist zwar im Produktiveinsatz in Einzelfällen zu erwarten, kann jedoch aufgrund der begrenzten Bearbeitungszeit nicht abgedeckt werden. Korrekt behandelt werden aber Fälle, in denen ein Append- und ein Write-Konflikt in derselben Zeile auftreten.

7.4.4 Benachrichtigung

Der Benutzer soll benachrichtigt werden, wenn in dem gerade bearbeiteten Outline durch Replikation ein Konflikt aufgetreten ist. Dies soll sofort nach der Replikation geschehen, ohne dass sich die Seite neu lädt oder der Benutzer in seinem Arbeitsfluss unterbrochen wird. Diese Anforderung wurde im Kapitel „Analyse“ (Abschnitt 3.4.2) gestellt. Erst wenn der Benutzer sich für eine Beschäftigung mit dem Konflikt entscheidet, muss er die Bearbeitung des Konflikts explizit starten. Die Umsetzung der Benachrichtigung wird in Abschnitt 8.6 genau beschrieben.

7.5 Benutzeroberfläche

In diesem Abschnitt wird der Einsatz der Technologien bei der Umsetzung der Benutzeroberfläche begründet und deren Architektur vorgestellt.

7.5.1 Strategien

Die Benutzeroberfläche soll mit einfachen HTML-Elementen umgesetzt werden. Dabei wird auf den Einsatz von komplexen Application Frameworks verzichtet; lediglich die JQuery-Bibliothek

wird eingesetzt, um die Entwicklung zu vereinfachen. Auch der Einsatz von Adobe Flash für die Umsetzung des Gliederungseditors wurde verworfen. Der Grund hierfür ist, dass ein DOM-Baum offenen Web-Standards entspricht, und daher den Anforderungen Erweiterbarkeit und Accessibility besser Folge geleistet werden kann.

Abgeänderte oder alternative Benutzeroberflächen sind für eine standardkonforme Architektur leicht umzusetzen. Beispielsweise könnte auf einfache Weise ein Screenreader implementiert werden, der auf Funktionstasten reagiert und nacheinander den Inhalt der Zeilen vorliest. Durch Einsatz von CSS lässt sich das Aussehen durch Veränderung der Stylesheets einfach anpassen. Menschen mit Sehschwierigkeiten können Kontraste und Schriftgrößen beliebig verändern. Ein Plugin zum Neuformatieren und Ausdrucken eines Outlines wäre ebenfalls denkbar; es müssen lediglich dass DOM traversiert und die einzelnen Elemente neu formatiert werden.

Mit beispielsweise einem Flash- oder Sproutcore-Widget [Spr10] wäre der Gliederungseditor als einzelnes Objekt auf der Seite eingebettet. Für andere Technologien wären die Inhalte somit nicht erreichbar. Die Ladezeiten können dadurch ebenfalls minimal gehalten werden, da die eingesetzten JQuery-Bibliotheken und der Anwendungscode nur einen geringen Umfang haben.

7.5.2 Seitenaufbau

Die Anwendung besteht aus drei Ansichten: Von einer Hauptübersichtsseite mit einer Auflistung aller Outlines kann in die Outline-Einzelansicht gewechselt werden, die gleichzeitig der Gliederungseditor ist. Die dritte Ansicht ist die Outline-Bearbeitungsansicht, auf der der Outline-Titel geändert sowie das Outline gelöscht werden kann.

Aktuelle Statusinformationen, also Hinweise, Fehler oder Erfolgsmeldungen, sollen auf der Seite integriert angezeigt werden. Dies soll nicht als Popup implementiert werden, um den Benutzer nicht in seinem Arbeitsfluss zu unterbrechen. Stattdessen soll beim Eintreten des entsprechenden Ereignisses ein HTML-Element ein- und durch eine JavaScript-Funktion nach kurzer Zeit wieder ausgeblendet werden.

Während auf Antwort der lokalen Datenbank gewartet wird, soll eine animierte Grafik (ein *Throbber*) anzeigen, dass das Programm eine Aktion ausführt. Dadurch erhält der Benutzer ein Feedback, eventuell auftretende Wartezeiten im niedrigen Sekundenbereich werden dadurch als nicht so störend empfunden [Mil68].

7.5.3 Editor

Das Kernstück der Anwendung ist der Gliederungseditor. Er besteht aus einer einzelnen Seite. Auf dieser befindet sich nicht ein einziges Texteingabe-Widget, wie bei einem klassischen Texteditor

zu erwarten wäre. Der Editor setzt sich vielmehr aus einer Vielzahl von einzelnen Textareas zusammen. Dadurch können die Zeilen als einzelne Formulare abgeschickt werden. Die Textareas können jederzeit editiert werden, zwischen ihnen kann mit Funktionstasten oder der Maus hin- und hergesprungen werden. Beim Verlassen einer Textarea wird ihr Inhalt sofort gespeichert. Der Baum von DOM-Elementen soll also dem Benutzer wie ein einzelnes Editor-Fenster erscheinen.

7.5.4 Interaktion

Beim Arbeiten mit dem Editor folgt die Interaktion mit dem System nicht dem für HTML-Dokumente üblichen Schema, nach dem eine Seite bei jedem Klick neu vom Server geladen wird. Stattdessen verändern sich bei den meisten Interaktionsvorgängen lediglich einzelne Elemente auf der Seite. Dafür wird das AJAX-Konzept eingesetzt. Die beiden Herangehensweisen wurden in Abschnitt 5.1.3.2 genauer erklärt, vgl. auch Abbildungen 5.2 und 5.3. Der Einsatz des AJAX-Konzeptes führt dazu, dass unter einer URL teilweise unterschiedliche Zustände von Ressourcen abgebildet werden. Dies widerspricht dem in Abschnitt 4.1.5 vorgestellten REST-Paradigma, nach dem jedem Zustand einer Ressource eine Adresse zugeordnet ist.

Das REST-Konzept wird aus gutem Grund durchbrochen. Wird zum Beispiel eine Zeile hinzugefügt, löst der entsprechende Tastendruck eine Veränderung im DOM sowie einen HTTP-Request aus. Die Manipulation des DOM ist jedoch minimal und kann deshalb durch eine einfache JavaScript-Funktion umgesetzt werden. Die Zeile muss außerdem sofort in der Datenbank persistiert werden. Zentral ist hier, dass die Veränderung im DOM unmittelbar eintritt, um ein flüssiges Arbeiten zu ermöglichen. Wenn der Benutzer jedesmal einen Server-Roundtrip abwarten müsste, bis er das Ergebnis seines Tastendrucks sieht, würde dies unverhältnismäßig lange dauern.

8 Systemdokumentation

In diesem Kapitel wird das fertige Ergebnis der Umsetzung des im letzten Kapitel vorgestellten Systems beschrieben. Zu Beginn wird die Struktur des Programmcodes der Anwendung erläutert. Eine Gesamtübersicht über das System wird mithilfe eines Fachklassendiagramms und einer Einführung in das Routing-Schema und die Datenstrukturen vermittelt. Des Weiteren werden die Module vorgestellt, die zur Umsetzung von Benutzeroberfläche, Replikation, Konflikterkennung und -behandlung entwickelt wurden. Danach wird auf die Umsetzung der Testsuite eingegangen.

In Abschnitt 5.2 wurde bereits das für das Deployment verwendete Cloud Computing vorgestellt; den Abschluss dieses Kapitels bildet eine Darstellung, wie die Anwendung mit Cloud Computing und den Amazon Web Services deployt wurde. Abschließend wird das Clustering Framework CouchDB-Lounge vorgestellt, mit dem das Deployment im Hinblick auf Skalierbarkeit optimiert wurde.

Im Rahmen dieser Arbeit kann nicht der gesamte Quelltext erklärt werden. Stattdessen werden die verschiedenen Ebenen des Systems vorgestellt. Nur technisch besonders schwierige oder signifikante Algorithmen oder Funktionen werden näher beleuchtet. Kürzere Quelltextauszüge finden sich zur Verdeutlichung im Text, längere im Anhang.

8.1 Projektstruktur

Um einen Überblick über die Struktur des Programmcodes zu vermitteln, wird in diesem Abschnitt der Inhalt der einzelnen Verzeichnisse des Projekts geschildert. Auf die einzelnen Klassen und Funktionen wird dann in den folgenden Abschnitten näher eingegangen. Die allgemeine Bedeutung der Verzeichnisse wurde bereits in den Abschnitten 4.2.5 und 5.1.1 erklärt.

_attachments: Hier liegen die Teile der Anwendung, die direkt im Browser ausgeführt werden können, sowie die Startseite (`index.html`).

app: Enthält die Fachklassen und die Datei `application.js`, in der das Routing Framework Sammy.js initialisiert wird. Hier werden alle Ressourcen aus den *Helpern* und Fachklassen geladen. In der Initialisierungsfunktion `init`, die hier aufgerufen wird, wird Verhalten an Fenster-, Maus- und Tastaturereignisse gebunden. Des Weiteren wird hier die Replikation gestartet.

_controllers: Aufgabe der *Controller* ist es, die in Abschnitt 8.2 vorgestellten Routen und ihr Verhalten zu definieren.

_helpers: In *Helpern* werden Funktionen ausgelagert, die keine Methoden von Fachklassen im engeren Sinne sind. Diese Funktionen sind hauptsächlich für Bereiche des Oberflächenverhaltens zuständig. Beispielsweise werden die in der `init`-Funktion verwendeten Tastaturereignisse hier definiert. Auch Funktionen zum Traversieren der Zeilen haben hier ihren Platz.

_lib: Hier sind selbstgeschriebene Bibliotheken untergebracht, die JavaScript um Funktionalität erweitern. In `resources.js` sind Funktionen abstrahiert, mit denen von den Controllern aus die Lese- und Schreibzugriffe auf der Datenbank vorgenommen werden können.

_models: Hier werden die „Klassen" `Outline`, `Note` und `NoteCollection` definiert. Es handelt sich um *Models* im Sinne von *Object/Relational Mapping*. Sie werden in Abschnitt 8.3 im Detail beschrieben. Außerdem werden hier die Funktionen zur Konflikterkennung, -präsentierung und -auflösung bestimmt.

_templates: Als HTML-Dateien sind hier die Templates für das Template Engine Mustache.js gespeichert. Aus diesen Partials werden die Seiten zusammengebaut.

_views: Es handelt sich hierbei nicht um CouchDB-Views, sondern um die Repräsentation der Models `Outline` und `Note` für die Fachlogik der Anwendung und zum Rendern. Bspw. eine `OutlineView` enthält ein `Outline`-Objekt und bereitet ihre Daten für die Mustache-Templates auf. Die Controller greifen nicht direkt auf die Models, sondern ausschließlich auf die View-Repräsentationen zu. Deutlich wird dies im Fachklassendiagramm in Abbildung A.9.

config: In die Konfigurationsdatei `config.js` können die URLs der Anwendung für die Replikation sowie der Name der Datenbank eingetragen werden. Im Unterverzeichnis **features** finden sich Konfigurationsdateien für die Testumgebung.

images: Kleine Grafiken, die für das Layout benötigt werden, liegen hier bereit.

spec: Hier sind die Unit Tests sowie das Unit Test Framework abgelegt. Sie werden in Abschnitt 8.8.1 beschrieben.

style: Enthält selbst erstellte und vom Blueprint Framework geerbte Stylesheets.

features: In diesem Verzeichnis sind die Integration Tests enthalten. Sie werden in Abschnitt 8.8.2 beschrieben.

filters: Hier liegen die Filterfunktionen, mit denen die Datenbank auf Änderungen und deren Konflikthaftigkeit überprüft werden kann.

Rakefile: Dieses enthält Makros, mit denen die Anwendung in bestimmte konflikthafte Zustände versetzt werden kann (Abschnitt 9.2.4).

README: Eine Kurzfassung der Installationsanleitung (Abschnitt 9.1) und der Bedienungsanleitung (Abschnitt 9.2).

vendor: Die eingebundenen Bibliotheken sind in diesem Verzeichnis gespeichert. Die einzelnen JavaScript-Dateien müssen jeweils in einem `_attachments`-Unterverzeichnis liegen, damit CouchDB sie ausführen kann.

views: Hier liegen die CouchDB-Views, mit denen die Daten aus der Datenbank aufbereitet angefordert werden können. Für die Anwendung werden Map-Funktionen aus insgesamt drei Views benötigt.

Die Abbildung in Abschnitt A.5.1 zeigt ein Fachklassendiagramm, das einen Überblick über die zentralen Fachklassen bietet.

8.2 Routing

Im Folgenden wird das URL-Schema der Anwendung beschrieben. Dadurch wird auch das URL-Schema einer CouchApp sowie einer Anwendung mit dem Routing Framework Sammy.js deutlich. Diese Technologien wurden in Abschnitt 5.1.1 bzw. 5.1.4 vorgestellt.

Die Startseite ist unter der URL `http://localhost:5984/doingnotes/_design/doingnotes/index.html#/` zu finden. Nach dem Server und dem Port, unter dem die Anwendung zu erreichen ist, wird der Name der Datenbank angegeben. Das Präfix `_design/` leitet ein Designdokument bzw. den Namen der Anwendung ein, die ebenfalls den Namen `doingnotes` trägt. Da die Datei `index.html` direkt im `_attachments`-Verzeichnis des Designdokuments gespeichert ist, kann sie im Designdokument aufgerufen werden. Der Schrägstrich nach dem HTML-Anker leitet auf die Sammy-Route mit dem Pfad `#/` weiter.

In den Controllern sind weitere Routen definiert, die im Folgenden kurz erklärt werden. Die Route mit dem Pfad `#/outlines` initialisiert eine neue `OutlinesView`, die eine Liste aller Outlines rendert. Der Aufruf von `#/outlines/new` rendert das Formular, mit dem ein neues `Outline` erstellt werden kann. Die Route `#/outlines/edit/:id` zeigt ein Formular, mit dem der Titel des Outlines geändert werden kann. Unter `#/outlines/:id` wird das Outline angezeigt, hier ist der Gliederungseditor zu finden. Die weiteren Routen haben `PUT`, `POST` oder `DELETE` zur Methode und sind demnach für die Benutzer transparent.

Die grundlegenden Datenbankoperationen *Create*, *Read*, *Update* und *Delete* wurden in dem von der Autorin erstellten Plugin `Resources.js` abstrahiert. So müssen diese immer wiederkehrenden Aufgaben nicht in jeder Route neu implementiert werden, sondern können teilweise für `Notes` und für `Outlines` wiederverwendet werden. In Listing A.1 ist ein Auszug aus dem Plugin zu finden: Darin werden u.a. die Methoden `new_object` und `load_object_view` defi-

niert. `new_object` nimmt den Typ des Objekts (z. B. `Outline`) und eine Callback-Funktion entgegen. Letztere wird ausgeführt, nachdem das Template für das entsprechende Objekt geladen wurde. `load_object_view` verlangt außerdem die ID des Objekts als Parameter. Das Dokument mit dieser ID wird von der Datenbank angefordert und ein View-Objekt dafür angelegt. Mit diesem kann z. B. ein Template gerendert werden, wie das folgende Beispiel, entnommen der Route `#/outlines/edit/:id`, zeigt:

```
load_object_view('Outline', '123', function(outline_view){
  context.partial('app/templates/outlines/edit.mustache', outline_view, function(
      outline_view){
    context.app.swap(outline_view);
  });
});
```

Listing 8.1: Rendern des Templates zum Bearbeiten eines Outlines

8.3 Datenstrukturen

Wie bereits in Abschnitt A.5.1 erläutert, handelt es sich bei den Fachklassen um JavaScript-Funktionen, die die Attribute als lokale Variablen speichern, die den Feldern der Datenbank entsprechen. Methoden werden implementiert, indem der Prototyp der Funktion um entsprechende Funktionen erweitert wird. Im Folgenden wird die Datenstruktur der Anwendung anhand des Aufbaus der CouchDB-Dokumente erläutert.

8.3.1 Outline

Ein `Outline` repräsentiert eine Datei, die im Gliederungseditor bearbeitet werden kann. Es enthält neben `_id` und `_rev` den Datentyp `Outline` sowie den Titel (`title`). Zeitstempel markieren die Erstellung (`created_at`) und die letzte Änderung (`updated_at`) des Dokuments. Letzterer wird erst angelegt, wenn der Titel des Dokuments geändert wurde. Die Zeitstempel werden mit dem Befehl `new Date().toJSON()` bei der Erstellung des Objekts erzeugt. Sie werden zur zeitlichen Sortierung der Outlines in der Outline-Übersicht verwendet.

```
{  "_id": "ce63ec5aaf501c567d200d89f200088a",
   "_rev": "2-00899e40fef865bb3fa294cd72860b8f",
   "created_at": "2010/07/04 12:12:52 +0000",
   "updated_at": "2010/07/04 12:28:39 +0000",
   "kind": "Outline",
   "title": "My Shopping List" }
```

Listing 8.2: Ein Outline-Dokument

8.3.2 Note

Ein `Note` repräsentiert die Zeile eines Outlines. Es enthält genau wie dieses die Felder `_id`, `_rev`, den Datentyp `Note`, `created_at` und `updated_at`. Die Zeitstempel werden hier für die Reihenfolge der Zeilen in einem Outline benutzt, wenn diese nach einer Replikation vom System festgelegt werden muss (s. Abschnitt 8.7.1).

In dem Feld `text` ist der Inhalt der Zeile gespeichert. `source` wird für die Benachrichtigung nach Replikation benötigt (s. Abschnitt 8.5). Mithilfe des Felds `outline_id` kann bestimmt werden, zu welchem Outline eine Zeile gehört. Die letzten drei Felder sind optional: `next_id` und `parent_id` werden zum Rendern der Baumstruktur in einem Outline benötigt. Dies wird in Abschnitt 8.4.3 näher erläutert. `first_note` ist ein Boolean und damit das einzige Feld, dessen Datentyp kein String ist. Es markiert die erste Zeile eines Dokuments, damit diese beim Traversieren leichter gefunden werden kann.

```
{
   "_id": "ce63ec5aaf501c567d200d89f2001b08",
   "_rev": "5-86d6c6ce0ad7b6b8454cbb91590e315c",
   "created_at": "2010/07/04 12:14:35 +0000",
   "updated_at": "2010/07/04 12:15:08 +0000",
   "kind": "Note",
   "text": "Hier ist Text der in einer Zeile eben so steht",
   "source": "eb8abd1c45f20c0989ed79381cb4907d",
   "outline_id": "ce63ec5aaf501c567d200d89f200088a",
   "next_id": "ce63ec5aaf501c567d200d89f2002a87",
   "parent_id": "ce63ec5aaf501c567d200d89f20015ab",
   "first_note": true
}
```

Listing 8.3: Ein Note-Dokument

8.4 Benutzeroberfläche

In diesem Abschnitt wird die Umsetzung der Benutzeroberfläche in Auszügen beschrieben. Eingegangen wird nacheinander auf die Implementierung des Gliederungseditors, die Operationen Speichern, Einfügen und Einrücken der Zeilen, deren Auswirkungen auf DOM und Datenbank, sowie das Rendern der Zeilen nach dem Neuladen des Outlines.

8.4.1 Umsetzung des Gliederungseditors

Im DOM ist der Gliederungseditor als ein `<div>`-Element dargestellt, das eine unsortierte Liste enthält (siehe Listing 8.4). Die `<li>`-Elemente der Liste entsprechen den Zeilen. Hat eine Zeile Kindknoten, also liegen unter ihr eingerückte Zeilen, wird in ihr `<li>` ein weiteres `<ul>` eingefügt, das wiederum weitere `<li>`-Elemente enthält. Das Ergebnis einer solchen Einrückungsoperation ist in Listing A.2 zu finden.

```
<div id="writeboard">
  <ul id="notes">
    {{#notes}}
      <li class="edit-note" id="edit_note_{{_id}}">
        <form class="edit-note" action="#/notes/{{_id}}" method="put">
          <span class="space"> </span>
          <a class="image"> </a>
          <textarea class="expanding" id="edit_text_{{_id}}" name="text">{{text}}</
              textarea>
          <input type="submit" value="Save" style="display:none;"/>
        </form>
      </li>
    {{/notes}}
  </ul>
</div>
```

Listing 8.4: Das Template für den Gliederungseditor in Mustache-Syntax

8.4.2 Modifizierung des DOM

Die Initialisierungsfunktion bindet bestimmtes Verhalten an die Textareas, die die Zeilen des Gliederungseditors repräsentieren. Wenn bestimmte Fenster-, Maus- und Tastaturereignisse eintreten, wird das Verhalten ausgelöst. Dadurch wird das Speichern, Einfügen und Einrücken der Zeilen umgesetzt.

Eine Zeile soll immer dann gespeichert werden, wenn der Maus-Fokus sie verlässt, egal, ob das durch eine Funktionstaste, die Maus oder das Schließen des Fensters geschieht. Sie soll dann nicht gespeichert werden, wenn sich der Zeileninhalt gegenüber dem in der Datenbank gespeicherten Text nicht verändert hat. Dies wird über ein Custom Data Attribute realisiert, wie es in Abschnitt 5.1.2 beschrieben wurde. Wenn in die Zeile hineinnavigiert wird, wird auf dem `NoteElement`, also der Repräsentation einer Zeile im DOM, die Methode `setDataText` aufgerufen. Darin wird der momentane Inhalt der Zeile gespeichert. Mit diesem Wert wird der Text beim Verlassen der Zeile verglichen. Sind die beiden Werte gleich, muss die Zeile nicht gespeichert werden.

Wird die Eingabetaste gedrückt, während sich der Fokus in einer Zeile befindet, wird in der Methode `insertNewNote` ein neues Note-Objekt erzeugt. In der Callback-Funktion wird das Partial für die neue Zeile mit den Werten gefüllt und mit einer jQuery-Methode in das DOM eingefügt. Außerdem werden mehrere Zeiger angepasst, wie bereits in Abschnitt 7.3.3.2 beschrieben: Die `next_id` der Zeile, an die die neue Zeile angehängt wurde, und ggf. auch die `parent_id`s der Nachfolger müssen den Modifikationen im DOM Folge leisten.

Ähnliches geschieht beim Ein- oder Ausrücken von Zeilen: Auch hier müssen die Zeiger von vor- und nachstehenden Zeilen sowie ggf. die des Elternknoten und seiner Nachfolger angepasst werden. Im ungünstigsten Fall zieht das Einrücken eines Knotens bis zu zwei weitere Schreibzugriffe nach sich. Im DOM wird beim Einrücken das `<li>`-Element der Zeile in ein `<ul>` gehüllt und in den neuen Elternknoten eingefügt.

8.4.3 Rendern der Baumstruktur

Im vorangegangenen Abschnitt wurde skizziert, was beim Interagieren mit dem Gliederungseditor mit dem DOM und der Datenbank geschieht. Wird ein Outline aber geöffnet oder neu geladen, müssen alle seine Zeilen auf einmal gerendert werden. Dazu werden alle Zeilen des Outlines auf einmal aus der Datenbank geladen. Dies wurde in Abschnitt 7.2.7 beschrieben. Im resultierenden Array der Zeilen wird die erste Zeile bestimmt; sie ist durch ein spezielles Attribut gekennzeichnet. Ausgehend von dieser wird mithilfe einer rekursiven Funktion der Baum traversiert. *Traversierung* bezeichnet „das Untersuchen der Knoten des Baums in einer bestimmten Reihenfolge“ [Knu97, Kap. 2.3]. Dabei werden alle Knoten systematisch untersucht, so dass jeder Knoten genau einmal besucht wird. Nach einer kompletten Traversierung liegt ein lineares Abbild der Knoten vor.

Die Funktion `renderNotes` erhält das Array mit allen Zeilen des Outlines und einen Zähler. Der Zähler hat zu Beginn den Wert der Länge des Arrays und wird bei jedem Durchlauf um eins dekrementiert. Außerdem wird die gerade untersuchte Zeile aus dem Array gelöscht. Nacheinander wird überprüft, ob die aktuelle Zeile einen Kindknoten oder einen Next-Zeiger hat. Wenn ja, wird diese Zeile ins DOM eingefügt und `renderNotes` erneut aufgerufen. Wenn eine Zeile weder Kindknoten noch Next-Zeiger besitzt, ist die letzte Zeile des Outlines erreicht und die Funktion ist beendet. Die Funktion ist in Listing A.3 dokumentiert.

In einem gerenderten Baum können Zeilen ein- und ausgeklappt werden. Das Einklappen der Kindknoten einer Zeile wird durch Ausblenden der Kindknoten realisiert. Außerdem wird das am Anfang der Zeile als Aufzählungszeichen eingeblendete Dreieick um 90 Grad gedreht. Das Einklappen einer Zeile wird nicht in der Datenbank persistiert.

8.5 Replikation

Die Funktionen, mit denen die Replikation gesteuert wird, sind im Plugin `ReplicationHelpers` enthalten. Sie werden in diesem Abschnitt vorgestellt.

8.5.1 Starten der Replikation

Durch den Aufruf der Funktionen `replicateUp` und `replicateDown` kann eine Continuous Replication zum bzw. vom Server gestartet werden. Die beiden Funktionen sind gleich aufgebaut, nur sind Quelle und Ziel gegengleich gesetzt:

```
1 replicateUp: function(){
2   $.post(config.HOST + '/_replicate',
3     '{"source":"' + config.DB + '", "target":"' + config.SERVER + '/' + config.DB + '",
          "continuous":true}',
4     function(){
5       Sammy.log('replicating to ', config.SERVER)
6     },"json");
7 }
```

Listing 8.5: Die Funktion `replicateUp`

Beide Funktionen werden in der Initialisierungsfunktion aufgerufen, wodurch die Replikation bei jedem manuellen Reload der Seite neu gestartet wird. Läuft sie bereits, wird der Befehl einfach ignoriert. Auf diese Weise kann die Replikation nach einer unterbrochenen Internetverbindung wieder aufgenommen werden. Die URLs und Ports von Client und Server werden in der Datei `config.js` angegeben.

8.5.2 Benachrichtigung bei Änderungen

In den Systemanforderungen wurde festgelegt, dass Benutzer über durch Replikation hervorgerufene Änderungen benachrichtigt werden sollen, ohne in ihrem Arbeitsfluss unterbrochen zu werden. Die Seite, in die der Gliederungseditor eingebettet ist, enthält ein Element, in dem die Replikationsmeldung steht. Das Element ist üblicherweise ausgeblendet, es wird nur beim Vorliegen von Änderungen sichtbar gemacht. Wird dann dem darin enthaltenen Link gefolgt, wird die Seite neu geladen und die neue Version des Outlines angezeigt.

```
1 <h3 style="display:none;" id="change-warning">Replication has brought updates. <a href
      ="javascript: window.location.reload();">View them.</a></h3>
```

Listing 8.6: Benachrichtigung über Änderungen

Um das Vorliegen von Änderungen festzustellen, wird bei jedem Speichern einer Zeile das Feld `source` neu gesetzt. Es erhält den Hash des Rückgabewerts von `window.location.host`. Dieser String ermöglicht es, die URL und den Port des Systems, auf dem die Zeile geändert wurde, eindeutig zu identifizieren. Durch das Hashen wird verhindert, dass personenbezogene Daten gespeichert werden.

Wenn ein Outline angezeigt wird, wird die Funktion `checkForUpdates` auf dem Outline aufgerufen. Darin werden der Changes-Feed abgefragt und mit dem Filter in Listing 8.7 alle Zeilen mit fremder `source` herausgefiltert. Wenn der Hash des eigenen Hostnamen „848c7“ ist, geschieht dies mit dem Aufruf von http://localhost:5984/doingnotes/_changes?filter=doingnotes/changed&source=848c7.

```
function(doc, req) {
  if(doc.kind == 'Note' && doc.source != req.query.source) {
    return true;
  }
  return false;
}
```

Listing 8.7: Der `changed`-Filter für den Changes-Feed

Dieser Aufruf hat zum Ziel, die Sequenznummer der Datenbank zum Zeitpunkt der letzten fremden Änderung zu bekommen, denn Änderungen sind erst ab diesem Zeitpunkt als neu zu bewerten (vgl. Abschnitt 4.2.4). Die Sequenznummer wird in der Variablen `since` gespeichert. Dann wird ein neuer XMLHttpRequest abgeschickt, der die Datenbank ab diesem Zeitpunkt auf Änderungen überwacht. Mit angegeben werden muss der `heartbeat`-Parameter, damit CouchDB in Intervallen der angegebenen Millisekunden Zeilenumbruch-Zeichen schickt. So wird verhindert, dass der Browser fälschlicherweise einen Timeout feststellt. Der `feed=continuous`-Parameter ermöglicht das Offenhalten der HTTP-Verbindung, so dass ähnlich wie bei Continuous Replication Änderungen fortlaufend gesendet werden.

Bei der Sequenznummer „142“ geht die Anfrage an die Adresse http://localhost:5984/doingnotes/_changes?filter=doingnotes/changed&source=848c7&feed=continuous\&heartbeat=5000&since=142. Sobald der übermittelte ResponseText eine Dokumentenrevision mit einem `changes`-Attribut enthält, wird dieses Dokument geöffnet und die oben beschriebene Benachrichtigungsmeldung eingeblendet. Die Funktion, die das hier beschriebene Verfahren umsetzt, ist in Listing A.4 dokumentiert.

Gegen die Benutzung des Changes-Feeds im `feed=continuous`-Modus spricht, dass momentan nur der Browser Firefox diese Option sicher unterstützt. Andere gängige Browser ignorieren diese Option oder reagieren mit fehlerhaften Ausgaben. Da sich dies mit zukünftigen Releases anderer Browser zu ändern verspricht, wurde diese Einschränkung in Kauf genommen.

8.6 Konflikterkennung

Die Datenbank muss nicht nur auf Änderungen, sondern auch auf Konflikte überwacht werden. Auch hier sollte der Benutzer unmittelbar benachrichtigt werden, damit Konflikte möglichst zeitnah gelöst werden können.

Das Modul `ConflictDetector` ist, ebenso wie der im nächsten Abschnitt vorgestellte `ConflictResolver`, als Singleton implementiert. Die Methode `checkForNewConflicts` benutzt, wie im vorangehenden Abschnitt beschrieben, den Changes-Feed im Continuous-Modus, um Änderungen zu überwachen. Die Filterfunktion wird hier benutzt, um konflikthafte Zeilen herauszufiltern. Wenn eine Zeile mit einem `_conflicts`-Array gefunden wird, und ihre `outline_id` mit der aktuell angezeigten Outline übereinstimmt, wird die erste konflikthafte Revision der Zeile geladen. Die Konfliktart wird identifiziert, und die beiden konkurrierenden Revisionen der Zeilen werden dem `ConflictResolver` bzw. dem `ConflictPresenter` zur weiteren Bearbeitung übergeben.

Wenn Konflikte nicht direkt gelöst werden, sollen sie auch nach einem erneuten Öffnen des Outlines angezeigt werden. Diese Konflikte werden deshalb vom `ConflictDetector` zwischengespeichert. Beim Öffnen des Outlines wird die Methode `checkForNewConflicts` aufgerufen; in dieser wird geprüft, ob die gespeicherten Konflikte zum aktuellen Outline gehören, und ggf. an den `ConflictPresenter` weitergereicht. Eine Hürde für die Umsetzung dieser Anforderung bestand darin, dass eine Änderung im `_conflicts`-Array eines Dokuments bislang keine Änderung für den Changes Feed von CouchDB darstellte. Daher musste zur Umsetzung dieses Features ein Patch für den Changes Feed geschrieben werden. Dieser Patch [Apa10h] ist in Abschnitt A.5.6 und in [Apa10a] dokumentiert.

8.7 Konfliktbehandlung

Im Prototypen der Anwendung werden zwei unterschiedliche Konfliktarten behandelt: Append-Konflikte und Write-Konflikte. Diese sind in Abschnitt 7.4 definiert. Im Folgenden wird die Umsetzung der Aufbereitung und Lösung beider Konfliktarten sowie deren Mischform beschrieben.

8.7.1 Append-Konflikte

Der `ConflictDetector` überprüft die beiden konkurrierenden Revisionen eines konflikthaften Dokuments auf die Art der Änderung. Unterscheiden sich die Revisionen in der `next_id`,

liegt ein Append-Konflikt in der darauffolgenden Zeile vor. Dieser Konflikt kann von der Anwendung selbst gelöst werden. Dazu werden die beiden Revisionen dem `ConflictResolver` übergeben. In der Methode `solve_conflict_by_sorting` lädt dieser die Zeilen, die den beiden konflikthaften Revisionen folgen. Diese werden zeitlich sortiert nacheinander in die Baumstruktur eingefügt. Als Resultat dieser Operation stehen sie also direkt untereinander. Die Zeiger in den umliegenden Zeilen werden ebenfalls so angepasst, dass sie den neuen Baum korrekt wiederspiegeln.

Auf der Benutzeroberfläche werden die Zeilen entsprechend ihrer Reihenfolge automatisch ins DOM eingebaut. Außerdem wird für einige Sekunden die Meldung „*Replication has automatically solved updates.*“ eingeblendet. Die veränderten Zeilen werden kurz hellgrün hinterlegt.

8.7.2 Write-Konflikte

Stellt der `ConflictDetector` fest, dass der Text der konkurrierenden Zeilen voneinander abweicht, handelt es sich um einen Write-Konflikt. Die beiden Revisionen werden an die Funktion `showWriteConflictWarning` im `ConflictPresenter` weitergereicht. Diese hinterlegt die konflikthaften Zeilen hellrot und blendet, ähnlich wie nach einem automatisch erfolgten Replikationsvorgang, eine Benachrichtigung ein:

```
1 <h3 style="display:none;" id="conflict-warning">Replication has caused one or more
    conflicts. <a href="/doingnotes/_design/doingnotes/index.html#/outlines/{{
    outline_id}}?solve=true">Solve them.</a></h3>
```

Listing 8.8: Benachrichtigung über konflikthaften Status der Datenbank

Die Benachrichtigung enthält einen Link zum aktuellen Outline mit dem Parameter `solve=true`. Wird das Outline mit diesem Parameter geladen, wird der `ConflictPresenter` wieder aktiviert. In der Methode `showConflicts` werden mithilfe einer CouchDB-View alle konflikthaften Zeilen für das aktuelle Outline geladen. Für jede Zeile werden wieder die beiden konkurrierenden Revisionen von der Datenbank angefordert. Dann wird im DOM die Textarea jeder konflikthaften Zeile durch ein Partial ersetzt, das den Text der beiden Revisionen für den Benutzer zur Auswahl anbietet. Die Textarea wird dabei durch zwei Formulare ersetzt, die jeweils den Text einer Revision enthalten. Jedes Formular enthält einen Speicherbutton, der einen `PUT`-Request auf die Route `#/notes/solve/:id` auslöst. Im Callback der Route wird der `ConflictResolver` angewiesen, mithilfe der Funktion `solve_conflict_by_deletion` eine neue Revision der Zeile anzulegen, die den vom Benutzer ausgewählten und evtl. manuell geänderten Text enthält. Diese Revision wird gespeichert und die beiden alten Revisionen werden gelöscht.

Sind mehrere Konflikte zu lösen, löst das Speichern einer der Versionen der Zeile aus, dass die beiden Formulare verschwinden und durch eine gewöhnliche Zeile mit Textarea ersetzt werden.

Mit dem Lösen des letzten Konflikts wird die Seite ohne den Parameter `solve` neu geladen. Damit wird auch die während des Lösens der Write-Konflikte ausgesetzte Überwachung nach Konflikten neu gestartet.

8.7.3 Kombination aus beiden Konfliktarten

Es kann der Fall auftreten, dass gleichzeitig der Text einer Zeile geändert und außerdem nach ihr eine Zeile eingefügt wird. In diesem Fall wird zuerst der Append-Konflikt wie oben beschrieben gelöst. Danach wird der Write-Konflikt künstlich erneut erzeugt. Dies wird über die `bulkSave`-Methode von CouchDB realisiert. Wird ein Dokument auf diese Weise und mit Angabe des Parameters `all_or_nothing:true` gespeichert, akzeptiert CouchDB die Änderung auch dann, wenn durch sie die Datenbank in einen konflikthaften Zustand gerät. Dann wird zusätzlich zu der Benachrichtigung über den gelösten Append-Konflikt auch der Hinweis über einen zu lösenden Write-Konflikt eingeblendet.

8.8 Systemtest

In den Abschnitten 5.3.1.2 und 5.3.2 wurden Testgetriebene Entwicklung sowie die Testing Frameworks JSpec und Cucumber dargestellt. Im Folgenden wird die Umsetzung einer Testsuite mit diesen Technologien beschrieben.

8.8.1 Unit Tests

Das in Abschnitt 5.3.2.1 vorgestellte Unit Test Framework JSpec wurde eingesetzt, um die Logik von Fachklassen und Helpern umfangreich zu testen. Die Tests sind je nach Modul und Funktionalität in mehrere Dateien aufgeteilt. In den Dateien sind Tests für folgende Funktionen enthalten:

note_element_spec: Traversierung und automatische Speicherung der Zeilen

inserting_note_element_spec: Einfügen der Zeilen

indenting_note_element_spec: Einrücken der Zeilen

unindenting_note_element_spec: Ausrücken der Zeilen

focusing_note_element_spec: Setzen des Fokus beim Navigieren im Gliederungseditor

rendering_note_element_spec: Darstellung der Baumstruktur beim Öffnen eines Outlines

outline_spec, outline_helpers_spec: Sortierung und Darstellung der Outlines

note_spec, note_collection_spec: Auffinden bestimmter Zeilen beim Rendern eines Outlines

resources_spec: Abstraktion der Datenbankoperationen

lib_spec: Erweiterungen für die Datentypen String und Array

conflict_spec: Darstellung der Zeile beim Lösen eines Write-Konflikts

Die Testsuite wird ausgeführt, indem die Datei `/_attachments/spec/index.html` im Browser geladen wird. Die Datei ist in Listing A.7 dokumentiert. In ihr werden zuerst die JSpec-Bibliotheken geladen: das Testing Framework und der Programmcode, der getestet werden soll oder für die Ausführung des zu testenden Codes benötigt wird. Danach wird die Funktion `runSuites()` definiert. Darin wird auf dem `JSpec`-Objekt, das durch die Einbindung der JSpec-Bibliothek vorhanden ist, die Methode `exec` jeweils einmal mit jeder der oben genannten Dateien als Parameter aufgerufen.

Zuletzt wird das Ausführen der Tests und die Ausgabe der Ergebnisse angestoßen. Dabei wird die Information über die Position der *Fixtures* mit angegeben. Fixtures sind Dateien, die HTML-Blöcke enthalten. Da die JSpec-Tests nicht auf der Datenbank operieren, sondern nur die Javascript-Funktionen testen, wird auf diese Weise das DOM in einem bestimmten Zustand simuliert. Im Verzeichnnis `/_attachments/spec/fixtures` liegen Fixtures, die Outlines in mehreren Zuständen sowie eine HTML-Repräsentierung der Startseite enthalten. Die Tests arbeiten mit diesen HTML-Seiten wie der Produktionscode mit dem generierten DOM.

Die Ausgabe der Testergebnisse erfolgt im Browser. In der Ausgabe enthalten sind ggf. Informationen über fehlerhafte Tests, die Anzahl der Tests sowie die Laufzeit, die in Abbildung 8.1 etwas über drei Sekunden beträgt.

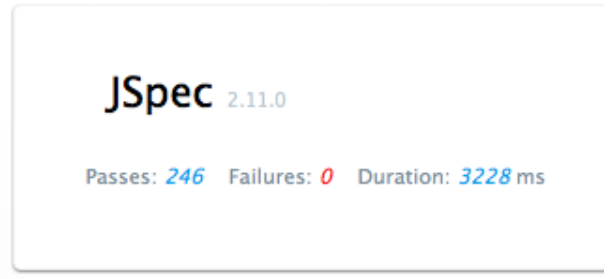

Abbildung 8.1: JSpec: Erfolgreiche Ausführung aller Unit Tests

8.8.2 Integration Tests

Das Framework Cucumber wurde bereits in Abschnitt 5.3.2.2 vorgestellt. Es wird üblicherweise mit der Bibliothek *Webrat* [Helo8] verwendet. Webrat implementiert einen Browser-Simulator,

der allerdings kein JavaScript beherrscht. Da die erstellte Anwendung komplett in JavaScript geschrieben ist, musste bei der Umsetzung der Integration Tests auf ein Setup aus *HTMLUnit* [Gar], *Culerity* [Bak] und *Celerity* [Lan] zurückgegriffen werden.

HTMLUnit ist eine Java-Bibliothek, die HTML parsen sowie JavaScript ausführen kann. HTMLUnit wird oft als ein *headless browser (kopfloser Browser)* bezeichnet [Lano9b], da es über die Fähigkeiten eines Browsers verfügt, aber keine Benutzeroberfläche hat, in der die Seiten dargestellt werden. Die von HTMLUnit gelesenen bzw. ausgeführten Webseiten werden also nirgends angezeigt. Für die Ausführung der Testsuite ist HTML mindestens in Version 2.7 erforderlich.

Celerity ist ein JRuby-Wrapper um HTMLUnit. Es bietet eine API für die am häufigsten verwendeten Browser-Funktionen, die dann auf HTMLUnit ausgeführt werden. Culerity ist ein Ruby-Gem, das Cucumber mit Celerity verbindet, auch wenn der Code nicht in einer JRuby-Umgebung ausgeführt wird. Bei der Verwendung mit Culerity erzeugt Celerity einen Java-Prozess, an den alle Celerity-Aufrufe weitergeleitet werden. Die Ergebnisse werden in der Ruby-Umgebung genau so ausgegeben, wie wenn sie von einem einzigen Ruby-Prozess erzeugt worden wären [Lano9b].

Culerity liefert eine Reihe häufig verwendeter Step-Definitionen mit. Diese liegen in der Datei `/features/step_definitions/common_culerity_steps.rb`. Die weiteren Step-Definitionen sind im selben Verzeichnis zu finden.

In Abschnitt 5.3.2.2 wurden bereits Beispiele für Features/Szenarios (s. Abschn. 5.8) und Step-Definitionen (s. Abschn. 5.9) angegeben. Diese sind der Testsuite für die Anwendung entnommen. Die weiteren Features sind im Ordner `/features` gespeichert. Der erfolgreiche Durchlauf aller Szenarios wird durch Abbildung 8.2 veranschaulicht. Die Laufzeit beträgt etwa eineinhalb Minuten. Getestet werden die Funktionen: Outline anlegen und löschen; Titel eines Outlines ändern; zeitliche Sortierung der Outlines in der Übersicht; Zeile einfügen, bearbeiten, ein- und ausrücken.

Culerity betrachtet eine Änderung im Anker der URL nicht als Seitenwechsel. Damit dies richtig wahrgenommen wird und auch Sammy-Routen mit Cucumber getestet werden können, muss bei einer Änderung des Teils der URL nach dem Anker explizit die entsprechende Route aufgerufen werden. Da dieser Eingriff in die Funktionsweise von Sammy jedoch die Rückwärtsfunktion des Browsers deaktiviert, wird das Verhalten nur in der Testumgebung überschrieben. Deswegen wird im jeweils ersten Step eines Szenarios mit `$browser.execute_script("setTestEnv();")` die in der Datei `test_environment.js` enthaltene Funktion `setTestEnv()` aufgerufen, die die Testumgebung entsprechend setzt. Auf diese Weise können die Integration Tests ausgeführt werden, ohne dass das Verhalten der Anwendung beeinträchtigt wird.

```
  Scenario: delete an outline
[INFO] Visit your CouchApp here:
http://127.0.0.1:5984/doingnotes_test/_design/doingnotes/index.html
    Given an outline with the title "Songs"
    And a first note with the text "Rambling" and the id "123" for "Songs"
[INFO] Visit your CouchApp here:
http://127.0.0.1:5984/doingnotes_test/_design/doingnotes/index.html
    And I save
    When I go to the start page
    And I follow "Songs"
    And I follow "Change title or delete this outline"
    And I press "Delete this outline"
    Then I should see "Outline deleted."
    And I should not see "Songs"
    And I should see "You have no outlines yet"

9 scenarios (9 passed)
122 steps (122 passed)
0m55.681s
```

Abbildung 8.2: Cucumber: Alle Tests laufen durch

8.8.3 Testsuite für die CouchDB HTTP-API

Die Anwendung wurde mithilfe der JavaScript-HTTP-API von CouchDB entwickelt. Diese API ist ein Wrapper um die grundlegenden Datenbankfunktionen, die den Entwicklern das Formulieren der Anfragen an die Datenbank erleichtert. Anfragen können als einfache JavaScript- bzw. jQuery-Methodenaufrufe erfolgen, ohne dass eigens ein XMLHttpRequest erzeugt werden muss. Da bislang keine Tests für die API vorlagen und Teile der API offensichtlich fehlerhaft waren, wurde eine Testsuite implementiert. Diese wird demnächst unter der Apache Lizenz veröffentlicht [Apa10f]. Beispiele aus API und Testsuite finden sich in Abschnitt A.5.8. Des Weiteren wurden Verbesserungen an der API vorgenommen [Apa10e, Apa10g].

Die API-Tests werden, genau wie die Testsuite für die erstellte Anwendung, im Browser ausgeführt. Da sie im Gegensatz zu der oben beschriebenen Testsuite auf die Datenbank zugreifen, müssen sie von dieser ausgeliefert werden. Die Dateien können also nicht einfach im Browser geöffnet werden, sondern müssen in das Verzeichnis `/share/www` der CouchDB-Installation gelegt und mit dieser kompiliert werden. So sind die Tests, ähnlich wie Futon, unter der URL `http://localhost:5984/_utils/spec/run.html` erreichbar.

8.9 Deployment mit Amazon Web Services

Die Anwendung soll nicht nur, wie in der Einleitung (Abschnitt 1.1) ausgeführt, „nach unten skalieren“, sondern ebenfalls eine hohe Performance bieten: Auch wenn eine große Anzahl von Benutzern ihre Outlines zum selben Zeitpunkt über einen Server sychronisieren möchten, soll dies ohne Latenzerhöhung stets möglich sein. Wenn die CouchDB-Instanz auf dem Server einmal zeitweise nicht verfügbar ist, soll die Verfügbarkeit trotzdem gegeben sein. Mit Amazon Web Services lässt sich dies problemlos umsetzen.

Im Rahmen der Entwicklung des Prototypen wurde deshalb evaluiert, wie ein Deployment mit der *Amazon Elastic Compute Cloud (EC2)* umgesetzt werden kann. Die technischen Hintergründe von Cloud Computing wurden bereits in Abschnitt 5.2 dargestellt. Im Folgenden soll ein Überblick über die Konfiguration einer über EC2 deployten Anwendung gegeben werden. Die Darstellung stützt sich auf eigene Recherchen sowie auf [Bau10, Kap. 4.1].

AWS ist der Sammelbegriff für alle Cloud-Computing-Angebote der Firma Amazon. Amazon hat starke saisonale Schwankungen in der Nachfrage nach seinen Angeboten. Deswegen wird der Großteil der erheblichen IT-Ressourcen die meiste Zeit über nicht genutzt. Das AWS-Angebot resultiert aus der Geschäftsidee, die freien Ressourcen gegen Entgelt zur Verfügung zu stellen, wenn sie temporär nicht für eigene Produkte benötigt werden.

Mit EC2 kann der Benutzer über Web Services virtuelle Server verwalten. Um einen solchen Server einzurichten, müssen eine Reihe von Schritten ausgeführt werden. Diese Schritte sind im Anhang (Abschnitt A.5.9) mithilfe von Screenshots genauer dokumentiert.

Nach dem Erstellen des Amazon-Accounts wird zunächst ein *Schlüsselpaar* generiert, mit dem eine Identifizierung gegenüber der EC2-Instanz möglich ist (s. Abb. A.10). Der öffentliche Schlüssel wird mit dem Amazon-Account assoziiert, der private verbleibt auf dem Rechner des Benutzers.

Auch muss eine *Security Group* definiert und konfiguriert werden (s. Abb. A.11). Jede EC2-Instanz gehört einer solchen Security Group an. Über diese werden die Sicherheitseinstellungen definiert. Mit dem Public Key aus dem vorherigen Schritt können einzelne Ports freigeschaltet werden, über die auf den Server zugegriffen werden kann.

Des Weiteren muss die *Availability Zone* ausgewählt werden, also die geographische Region, auf der der Server laufen soll. Bei großen Installationen ist eine Verteilung auf unterschiedliche Zonen vorteilhaft, um gegen den Ausfall einer Region abgesichert zu sein. Mit einer optimalen Availability Zone kann außerdem die Latenz gering gehalten werden.

Nun wird die *Größe der Ressourcen* des Servers festgelegt (s. Abb. A.12). Es stehen verschiedene Pakete zur Verfügung, die sich in der Leistungsfähigkeit des Prozessors und in der Größe des Arbeitsspeichers sowie der Festplatte unterscheiden. Das Spektrum des Angebots erstreckt sich von

1,7 GB RAM / 160 GB Festplattenspeicher bis hin zu 68,4 GB RAM / 1690 GB Festplattenspeicher [Amaa].

Zuletzt muss ein *Amazon Machine Image (AMI)* ausgewählt werden (s. Abb. A.13). Ein AMI ist virtuelles Image, also eine Art Blaupause eines virtuellen Servers. Die AMIs unterscheiden sich bezüglich des Betriebssystems und der auf ihnen installierten Software-Pakete. Eigene Images können zur späteren Wiederverwendung angefertigt und auch gegen oder ohne Entgelt veröffentlicht werden. Für Testzwecke genügt es, den virtuellen Server aus einem vorgefertigten AMI zu erstellen. Gewählt wird eine aktuelle Ubuntu-Distribution, „alestic's 64bit server Ubuntu 9.04 AMI" mit der ID „ami-ccf615a5".

Die EC2-Instanz wird mit den oben festgelegten Parametern gestartet. Der neu entstandene virtuelle Server erhält automatisch eine öffentliche IP, unter der er im Internet erreichbar ist, und eine private, über die er mit anderen Instanzen kommunizieren kann. Diese IPs werden bei jedem Start neu vergeben. Deswegen ist zu empfehlen, eine *Elastic IP* einzurichten (s. Abb. A.14). Wird diese statische IP mit dem Server verknüpft, hat er nach jedem Neustart immer wieder dieselbe IP.

Die Verwaltung der Instanz erfolgt im Allgemeinen über die *AWS Management Console* (s. Abb. A.16). Diese ist unter der Adresse `https://console.aws.amazon.com/ec2/home` erreichbar. Die Verwaltung kann auch über Kommandozeilenbefehle erfolgen. Ist die Instanz bspw. unter der IP `184.73.233.128` erreichbar, und ist der Public Key in der Datei `.ssh/doingnotes.pem` abgelegt, ist ein Login in die Instanz mit dem Befehl `ssh -i .ssh/doingnotes.pem root@ec2-184-73-233-128.compute-1.amazonaws.com` möglich. Ist der Server eingerichtet, wird CouchDB wie auf einem normalen Ubuntu-Rechner installiert (siehe Anleitung in Abschnitt 9.1).

Wird die Instanz einmal beendet, werden alle Einstellungen und Installationen, die darauf vorgenommen wurden, gelöscht. Um Änderungen auch über die Laufzeit eines virtuellen Servers hinweg zu persistieren, muss der Zustand der Instanz extern gespeichert werden. Dafür kann *Amazon Elastic Block Store (EBS)* verwendet werden (s. Abb. A.15). Ein EBS wird nach dem Erstellen wie eine externe Festplatte als Laufwerk mit der EC2-Instanz verknüpft. Darauf können Momentaufnahmen des EC2-Servers gespeichert werden.

Die Kosten für eine EC2-Instanz richten sich nach der Leistungsfähigkeit des Servers und werden stundenweise abgerechnet. Der Preis setzt sich aus der Größe der Ressourcen, dem Volumen der Datentransfers und der Zeit der Verwendung von Elastic IPs und EBS zusammen. Unter [Amab] lassen sich die Kosten für das gewünschte Paket im Voraus berechnen.

8.10 Clustering mit Couchdb-Lounge

Im folgenden Abschnitt wird beschrieben, wie die Verteilung eines Systems auf mehrere CouchDB-Instanzen möglich ist, ohne dass sich für Benutzer des Systems etwas ändert. Für diesen Zweck kommt CouchDB-Lounge [Lee10] zum Einsatz. CouchDB-Lounge ist eine Proxy-basierte Anwendung für *Clustering* und *Partitionierung* [Lee09b]. Diese beiden Konzepte können helfen, die Verfügbarkeit bzw. die Leistung eines Systems zu erhöhen.

Clustering wurde bereits im Jahr 1997 als Lösungsansatz empfohlen, den steigenden Anforderungen von modernen Datenbankanwendungen nachzukommen [Meh97]. Ein Computer-Cluster bezeichnet allgemein eine Anzahl ...

> ... ähnlicher Arbeitsstationen oder PCs, die mithilfe eines lokalen Hochgeschwindigkeitsnetzwerkes miteinander verbunden sind. [Tan07, S. 34]

Im konkreten Fall beschreibt Clustering den redundanten Einsatz mehrerer CouchDB-Server, um *Lastverteilung (Load-Balancing)* und erhöhte Verfügbarkeit zu ermöglichen. Mehrere CouchDB-Instanzen laufen parallel; Anfragen werden gleichmäßig auf die Instanzen verteilt. Daten redundant zu speichern, damit im Fall eines Hardware-Ausfalls immer mehrere Kopien der Daten bereitstehen, kann ebenfalls mit CouchDB-Lounge umgesetzt werden. Dies wird im Rahmen dieser Arbeit jedoch nicht detaillierter ausgeführt.

Horizontale Partitionierung ist die Praxis, den Speicherplatz auf Partitionen aufzuteilen. Diese werden als *Shards* bezeichnet. Die Shards werden auf Server verteilt, um den Durchsatz zu erhöhen. Dadurch wird verhindert, dass die Performance der Festplatten zum Bottleneck wird. Im nächsten Abschnitt wird dies näher beschrieben.

8.10.1 Funktionsweise

CouchDB-Lounge ist eine neue Technologie, weder ihre Funktionsweise noch ihr Einsatz sind bisher gut dokumentiert. Die folgende Darstellung bezieht sich daher im Wesentlichen auf [And10, Kap. 19], und [Lee09a].

Lounge besteht aus zwei Hauptkomponenten. Ein *Smartproxy* behandelt CouchDB-Views und verteilt sie auf die anderen Knoten im Lounge-Cluster. Die Performance der Views kann demnach durch eine Erhöhung der Anzahl der Knoten im Cluster gesteigert werden. Der Smartproxy ist als Daemon für *Twisted* implementiert, ein „framework for writing asynchronous, event-driven networked programs in Python“ [Lef02]. Ein *Dumbproxy* ist ein Modul für *nginx*, ein Webserver und Reverse Proxy Server. Der Dumbproxy wird eingesetzt, um GET- und PUT-Requests entgegenzunehmen, die nicht an CouchDB-Views gerichtet sind. Auch diese Requests werden auf die einzelnen Knoten verteilt. Dabei wird den Clients der Eindruck vermittelt, es

handle sich um eine einzige CouchDB-Installation. Beide Module arbeiten mit eindeutigen Hashes, die CouchDB-Lounge aus den IDs der CouchDB-Dokumente bildet. Anhand der ersten Zeichen des hexadezimal dargestellten Hashes wird der Shard ausgewählt, dem dieses Dokument zugewiesen wird. Die genaue Zuteilung wird in einer *Shard-Map* konfiguriert.

Mit CouchDB-Lounge kann also ein Cluster erstellt werden, das nach außen unter einer Adresse erreichbar ist. Der Cluster wird in Abbildung 8.3 als Ring dargestellt. Den einzelnen der acht Knoten sind jeweils zwei Shards zugeteilt, denen wiederum jeweils eine Ziffer aus dem Hexadezimalsystem zugewiesen ist. Beginnt der Hash einer ID eines CouchDB-Dokuments mit dieser Ziffer, speichert Lounge dieses Dokument in den entsprechenden Shard. Durch das Verfahren werden HTTP-Requests zu dem tatsächlichen Speicherort des Dokuments weitergeleitet. So kann eine Partitionierung umgesetzt werden.

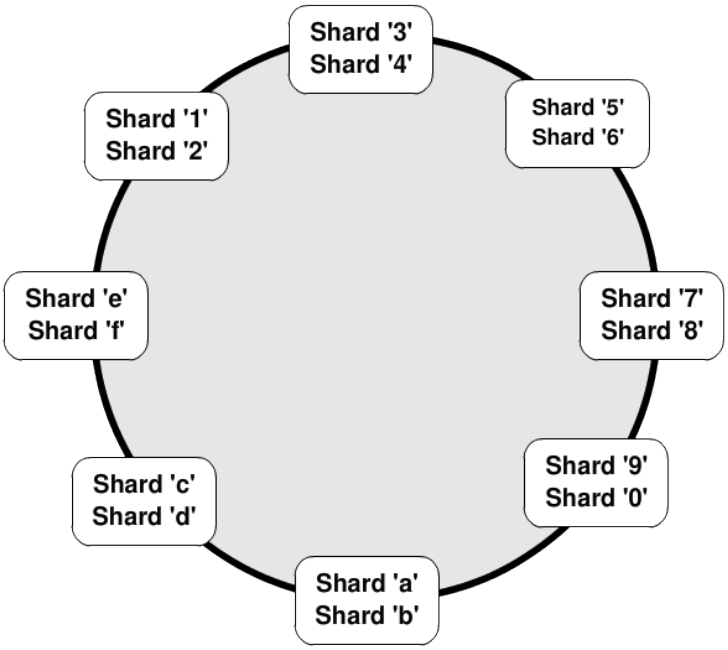

Abbildung 8.3: 16 Shards, 8 Knoten. Nach [Leh09]

> From the outside world, a couchdb lounge cluster looks just like any other couchdb node. [...] There's no difference from a functional perspective. [...] Its sharded nature is completely transparent. [VG10]

Ist zu erwarten, dass die Anforderungen an die Kapazität des Systems während seiner Lebensdauer steigen, wird in [And10] und [Kla10] empfohlen, die Anzahl der Shards zu Beginn möglichst hoch zu wählen. Die Daten auf mehr Shards zu verteilen, als Knoten verfügbar sind, wird „Oversharding“ genannt. Auch wenn eine geringe Anzahl Knoten zu Beginn ausreicht, kann diese später beliebig erhöht werden, indem weitere Knoten hinzugefügt werden. Die Shards werden dann

auf diese verteilt. Dies wird in Abbildung 8.3 skizziert. Das System in Abbildung 8.3 hat seine Schnittstelle behalten, die CouchDB-Installationen auf den Knoten wurden allerdings jeweils durch Lounge-Konfigurationen ersetzt. Sollen weitere Shards nachträglich hinzugefügt werden, muss der gesamte Cluster neu aufgebaut werden.

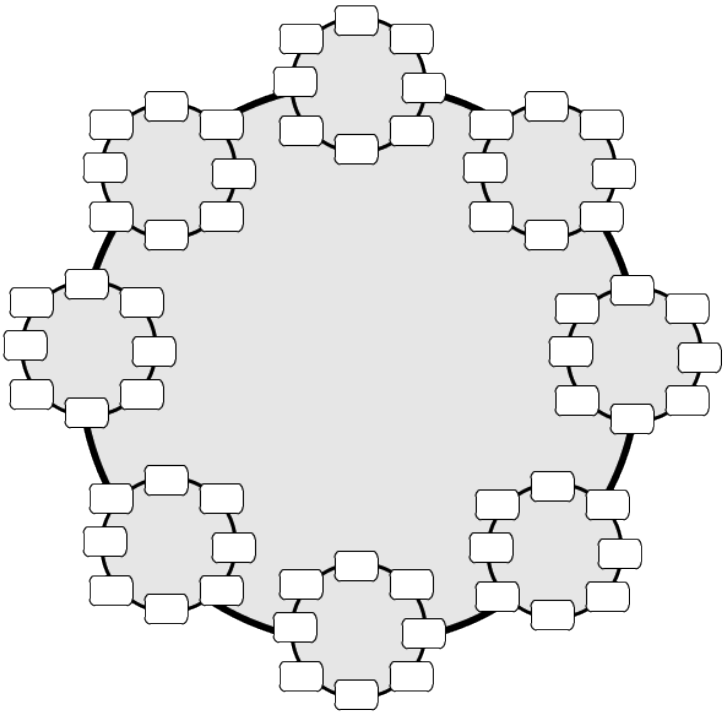

Abbildung 8.4: 16 Shards, 8 Knoten mit je 16 Sub-Shards, 8 Subknoten. Nach [Leh09]

Auch für kleinere Projekte besteht bei Oversharding der Vorteil, dass aufgrund der geringeren Anzahl der Dokumente die Größe des Index klein bleibt, und alle Operationen dadurch schneller ausgeführt werden können.

8.10.2 Konfiguration

In [Lee10] findet sich der Quelltext von CouchDB-Lounge. Er enthält Anweisungen, wie Dumbproxy, Smartproxy und *Python-Lounge*, eine Sammlung von benötigten Modulen, installiert werden müssen. In der aktuellen Version von CouchDB-Lounge besteht eine Abhängigkeit zu CouchDB in der Version 0.10.0. Nur für diese Version liegt ein Patch vor, der die benötigte „design-only replication“ aktiviert [Lee09a]. In späteren Versionen von CouchDB wird dieses Feature aber bereits vorinstalliert sein.

In [cou10a] ist beschrieben, wie auf einem Rechner mehrere CouchDB-Instanzen installiert werden können. Die zentrale CouchDB-Konfigurationsdatei in `/etc/couchdb/local.ini`

muss so oft kopiert werden, wie Instanzen laufen sollen. Listing 8.9 beinhaltet die wichtigen Teile aus der Kopie in `local-1.ini`. In den Kopien in `local-2.ini` etc. müssen die Angaben zu Port und Logfile entsprechend angepasst werden.

```
[httpd]
port = 5984
bind_address = 127.0.0.1

[log]
file = /var/log/couchdb/couch-1.log
```

Listing 8.9: Auszug aus der CouchDB Konfigurationsdatei

Listing 8.10 zeigt, wie die erste CouchDB-Instanz auf einem Unix-System (Mac OS X 10.6) gestartet werden kann:

```
sudo -i -u couchdb '/usr/local/bin/couchdb -a etc/couchdb/local-1.ini -p /usr/local/
    var/run/couchdb/couchdb-1.pid -o /usr/local/var/log/couchdb/error-1.log -e /usr/
    local/var/log/couchdb/error-1.log -b'
```

Listing 8.10: Starten einer CouchDB-Instanz

Ist die gewünschte Anzahl an CouchDB-Knoten eingerichtet und gestartet, wird die Funktionsfähigkeit des Setups überprüft, indem die CouchDB-Testsuite wie in der Installationsanleitung angegeben ausgeführt wird. Läuft die Testsuite fehlerfrei durch, muss CouchDB-Lounge vor der Benutzung konfiguriert werden. Dies geschieht durch Anpassung der Datei `/var/lounge/etc/shards.conf`. Darin wird die Anzahl der Shards und der Grad der Redundanz definiert. Die Datei enthält das JSON-Objekt `nodes`, in dem Informationen über die Anzahl der CouchDB-Knoten gespeichert sind. Jeder Eintrag in dem Array enthält den Hostnamen und den Port eines Knoten. Die `shard_map` ist ein Array aus Arrays, in dem bestimmt wird wo sich ein Shard befindet und wohin es repliziert werden soll. Die Anzahl der Shards und Knoten kann beliebig hoch gesetzt und beliebig redundant ausgelegt werden.

Listing 8.11 beschreibt zwei Shards auf zwei Knoten, von denen das erste Shard (mit der Nummer 0) sich auf Knoten 0 befindet, das zweite (mit der Nummer 1) sich auf Knoten 1. Das erste wird zu Knoten 1 repliziert, wenn Knoten 0 ausfallen sollte, und umgekehrt.

```
{
  "shard_map": [[0,1], [1,0]],
  "nodes": [ ["localhost", 5984], ["localhost", 5985] ]
}
```

Listing 8.11: shards.conf mit zwei Knoten und einfacher Redundanz

Listing 8.12 definiert einen Cluster mit acht Shards auf vier Knoten ohne Redundanz. Die Knoten auf beiden Beispielen liegen auf demselben Rechner, sie könnten aber auch durch Angabe eines anderen Hostnamen auf mehrere Maschinen verteilt werden.

```
{
  "shard_map": [[0], [1], [2], [3], [0], [1], [2], [3]],
  "nodes": [ ["localhost", 5984], ["localhost", 5985], ["localhost", 5986], ["
      localhost", 5987]] ]
}
```

Listing 8.12: shards.conf mit vier Knoten ohne Redundanz mit einfachem Oversharding

CouchDB-Lounge ist erfolgreich installiert und konfiguriert, wenn ein auf einem Knoten erstelltes Dokument automatisch auf all die Knoten kopiert wird, für die in `shard_map` Redundanz definiert ist.

9 Anwendung

Dieses Kapitel erklärt die Benutzung der im Rahmen dieser Arbeit erstellten Anwendung, die während der Entwicklung provisorisch *Doingnotes* genannt wurde. Hier sind Installation und Bedienung aller Funktionen dokumentiert.

9.1 Installation

In den folgenden beiden Abschnitten wird die Installation von CouchDB sowie das Deployment der Anwendung beschrieben.

9.1.1 CouchDB

Zuerst muss auf allen Rechnern, auf denen Doingnotes eingesetzt werden soll, die Datenbank CouchDB installiert werden. CouchDB ist ausführlich beschrieben in Abschnitt 4.2. Die Mindestversion ist 0.11.0. Frühere Versionen unterstützen noch nicht die für Doingnotes wichtige Continuous Replication.

CouchDB läuft auf allen gängigen Desktop-Betriebssystemen. Von den verbreiteten mobilen Plattformen werden zum jetzigen Zeitpunkt Google Android ([Mil10]) und Nokia MeeGo (ehemals Maemo) ([Ape09], [Ape10]) unterstützt. Vor kurzem kündigte die Firma Palm an, dass die nächste Version von Palm WebOS ebenfalls mit einer CouchDB-Installation ausgeliefert wird ([Pro10]).

Für einige Desktop-Betriebssysteme gibt es bereits vorkompilierte CouchDB-Pakete. In ihnen sind alle Abhängigkeiten enthalten. Diese Abhängigkeiten müssen bei einer Installation aus dem Quelltext selber aufgelöst werden. CouchDB benötigt u. a. Installationen von Erlang [Eri10], OpenSSL [Ope09] und Spidermonkey [Moz10c]. Genauere Informationen finden sich in Abschnitt 6.2.1.

Für Mac OS X ist der schnellste Weg an eine lauffähige CouchDB-Installation zu gelangen der Download von *CouchDBX*: „The one-click CouchDB package for the Mac“ [Leh10d]. Für Windows-Systeme gibt es ebenfalls einen Binary Installer [Cou10b]. Manche Linux-Distributionen haben CouchDB in ihre Software Repositories aufgenommen; beispielsweise in neueren Ubuntu-Versionen kann CouchDB mit dem Paketsystem *apt* installiert werden.

Die aktuellen Versionen der drei vorgestellten Binaries sind ausreichend für den Einsatz von Doingnotes und für ein einfaches Nutzen der Anwendung durchaus zu empfehlen. Für Entwickler empfiehlt es sich aber, CouchDB direkt aus dem Quelltext zu installieren. Dabei kann die aktuellste Version aus dem Subversion [Apa10d] oder Git [Apa10c] Repository verwendet werden. Eine genaue, aktuelle Anleitung für viele Betriebssysteme und Distributionen findet sich im CouchDB Wiki [Cou10c]. Das Starten von CouchDB erfolgt betriebssystemabhängig, deshalb sei hier ebenfalls auf das Wiki verwiesen.

9.1.2 Deployment

Der nächste Schritt ist CouchApp [Che10] zu installieren. CouchApp wurde in Abschnitt 5.1.1 vorgestellt. CouchApp ermöglicht das einfache Deployment der Anwendung in die CouchDB-Instanz. CouchApp setzt eine aktuelle Python-Installation voraus [Pyt10]: Das Python-Modul `easy_install` [Felo9] wird benutzt, um das Python-Paket CouchApp herunterzuladen und zu installieren.

Um die Designdokumente (also den Programmcode) in die CouchDB-Instanz zu deployen, muss die `.couchapprc`-Datei im Projektverzeichnis angepasst werden; der Eintrag `default` muss auf die laufende CouchDB-Instanz zeigen (s. Listing 5.1). Anschließend muss von dort aus `couchapp push` aufgerufen werden, um die Anwendung lauffähig in die Datenbank zu deployen.

Für den Betrieb ist ein Browser nötig, der die in Abschnitt 8.5.2 diskutierten Anforderungen erfüllt. Es wird Firefox [Moz10d] ab Version 3.5 empfohlen.

9.2 Bedienung

9.2.1 Grundfunktionen

Die Anwendung ermöglicht das geordnete Speichern von Zeilen innerhalb von Outlines. Die geöffnete Seite `http://localhost:5984/doingnotes/_design/doingnotes/index.html` ist die Hauptübersichtsseite. Sie enthält wie alle Seiten eine Navigationsleiste und einen Inhaltsbereich. Die Hauptübersicht kann von überall aus durch einen Klick auf den Link „Outlines“ erreicht werden. Im Inhaltsbereich sind die bestehenden Outlines aufgelistet (s. Abb. 9.1). Neben dem Titel jedes Outlines stehen Datum und Uhrzeit der Erstellung. Sind noch keine Outlines vorhanden, wird der Text „You have no outlines yet.“ eingeblendet. Auf der rechten Seite finden sich Hinweise zum Editieren eines Outlines.

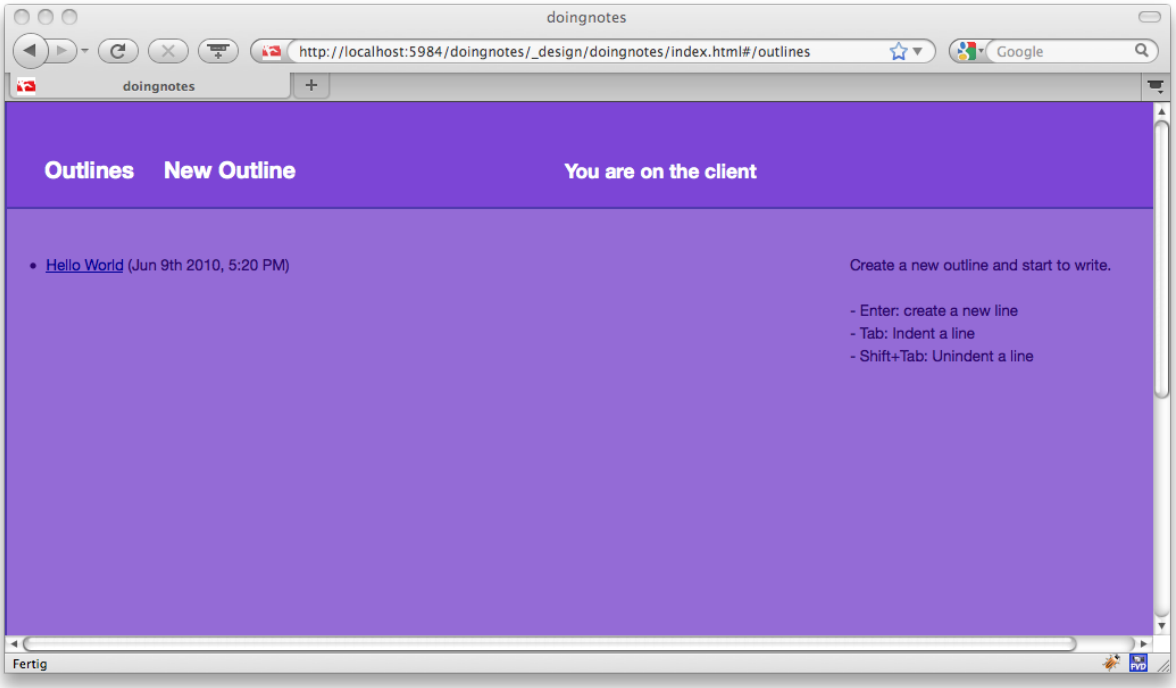

Abbildung 9.1: Screenshot: Liste der Outlines

Um ein neues Outline zu erstellen, muss dem Link „New Outline“ gefolgt werden. Titel müssen mindestens drei Zeichen enthalten und dürfen nur aus Buchstaben, Zahlen, Leerzeichen und dem Zeichen „-“ bestehen. Der Titel muss nicht eindeutig sein, ein Outline wird über die ID referenziert. Bei Fehleingaben sowie bei Hinweisen zu Replikation und Konflikten werden Benachrichtigungen eingeblendet. Diese erscheinen am oberen Rand der Anwendung und werden nach einigen Sekunden wieder ausgeblendet. Die Benachrichtigungen sind in Signalfarben gehalten: Fehler werden in rot, Hinweise in grün dargestellt.

Ein neu erstelltes Outline (s. Abb. 9.2) öffnet sich sofort. Der Text wird in die erste Zeile eingegeben. Wenn der eingegebene Text länger als die Zeile ist, wächst die Zeile automatisch mit. Mit der Eingabe von **Enter** wird eine neue Zeile erstellt. Ein drehender Throbber in der rechten oberen Ecke zeigt an, dass der Erstellungsprozess der Zeile noch in Gang ist.

Zwischen den erstellten Zeilen wird mit den **Pfeiltasten** gewechselt. Mit der **Tab**-Taste kann eine Zeile nach rechts eingerückt werden, um eine Hierarchisierung der Einträge zu realisieren. Eine Zeile kann so lange eingerückt werden, wie eine Zeile der nächsthöheren Ebene direkt über ihr steht. Mit **Shift** + **Tab** wird eine Zeile wieder ausgerückt. Bei Zeilen in der ersten Ebene hat diese Tastenkombination keine Wirkung. Beim manuellen Lösen eines Konflikts (s. Abschnitt 9.2.3) kann in den Konfliktfeldern ebenfalls mit **Tab** oder **Shift** + **Tab** hin- und hergesprungen werden.

Der Link „Change title or delete this outline“ führt auf die Bearbeitungsseite des Outlines (s. Abb.

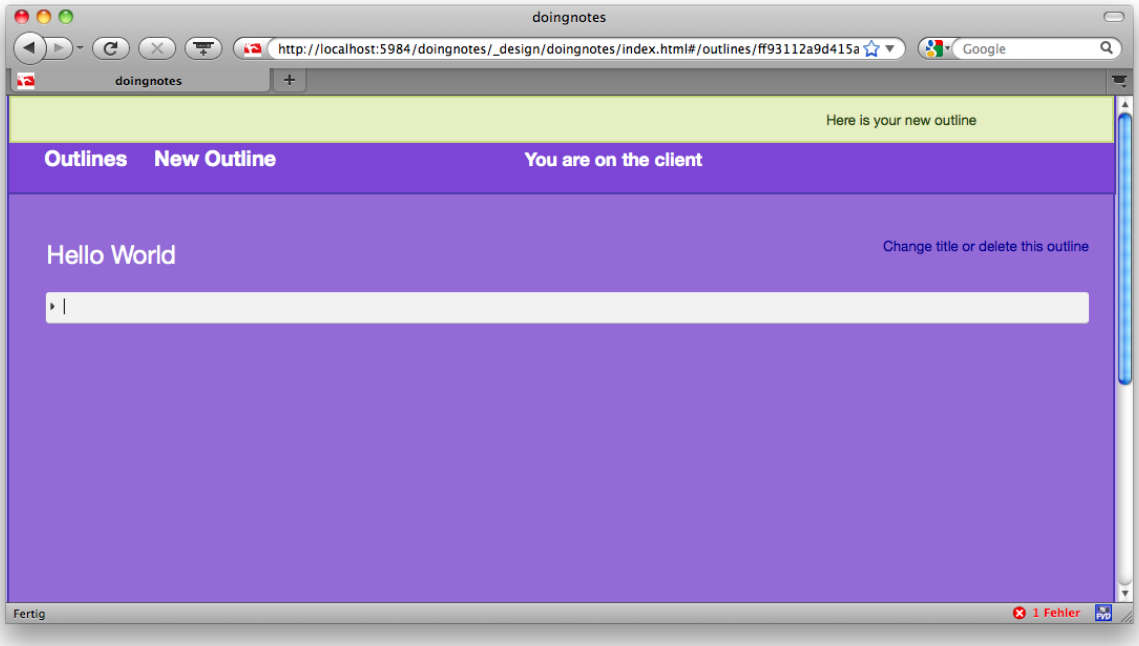

Abbildung 9.2: Screenshot: Neu erstelltes Outline

9.3). Dort kann der Titel des Outlines geändert werden. Mit einem Klick auf den Button „Delete this outline“ wird das Outline ohne Rückfrage gelöscht.

Nachdem der Benutzer nun mit der Bedienung der Anwendung als Single-User-System vertraut ist, wird in den folgenden beiden Abschnitten der Umgang mit den Multi-User-Features vorgestellt.

9.2.2 Replikation

Damit Replikation und Konfliktbehandlung zum Einsatz kommen können, muss Doingnotes in mehr als einer CouchDB-Instanz laufen. Für den vollständigen Betrieb von Doingnotes ist es deshalb nötig, es auf einen Server zu deployen. Die CouchDB-Instanz auf dem Server wird im Folgenden als „Server“ bezeichnet, die Instanz, auf der die Endnutzerin arbeitet, als „Client“.

Server und Client können durchaus auf demselben Rechner laufen. Um die Replikation zu testen, muss ein und dasselbe Outline gleichzeitig in mehr als einem Browser-Fenster geöffnet werden. So kann beobachtet werden, wie Updates auf dem Server (oder auf anderen Clients) automatisch zum Client repliziert werden, und der Konfliktlösungsmechanismus gegebenenfalls anspringt. Ob sich der Server nun auf demselben Rechner befindet oder nicht, seine URL muss in der Konfigurationsdatei `/_attachments/app/config/config.js` richtig eingetragen werden.

Für den Betrieb auf einem Rechner müssen zwei CouchDB-Instanzen installiert werden. Dies ist

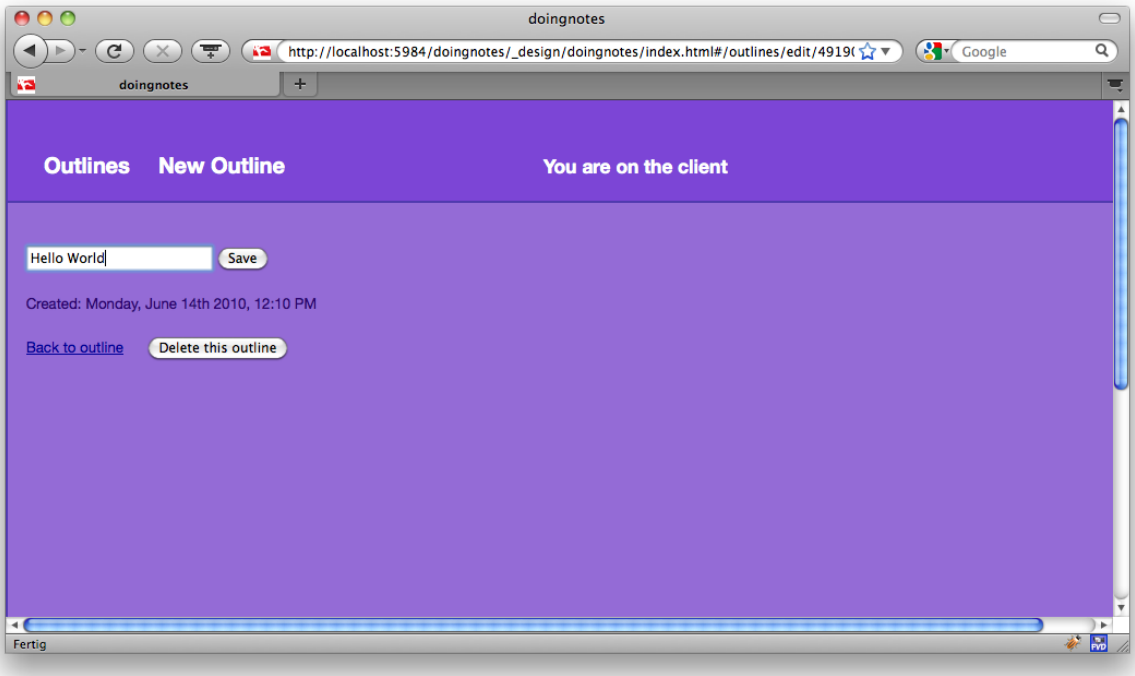

Abbildung 9.3: Screenshot: Ändern des Outline-Titels

in Abschnitt 8.10.2 beschrieben, siehe auch [cou10a]. Mit diesem Setup muss die Server-URL in `/_attachments/app/config/config.js` auf `http://localhost:5985` gesetzt werden.

Die CouchDB-Instanzen auf den Ports 5984 und 5985 (Client und Server) werden in zwei Browser-Fenstern geöffnet. In der Navigationsleiste findet sich für eine bessere Übersichtlichkeit ein Hinweis, in welchem Browser-Fenster man sich befindet. Angezeigt wird entweder „You are on the client“ oder „You are on the server“. In beiden Fenstern wird nun dasselbe Outline geöffnet. Wenn sich in dem Server-Fenster etwas ändert, wird im Client-Fenster ein Hinweis eingeblendet: „Replication has brought updates.“ (s. Abb. 9.4). Durch einen Klick auf „View them.“ wird die Seite neu geladen und das Update erscheint (s. Abb. 9.5).

Arbeitet man zwischenzeitlich offline oder bricht die Netzwerkverbindung ab, kann die Replikation durch ein einfaches Neuladen der Seite wieder aufgenommen werden.

9.2.3 Konfliktbehandlung

Wenn durch die stattgefundene Replikation kein Konflikt erzeugt wurde, wird das Ergebnis der Replikation einfach angezeigt. Es gibt zwei Arten von Konflikten, die von der Anwendung behandelt werden: *Append-Konflikte* und *Write-Konflikte* (vgl. Abschnitte 7.4.1 und 7.4.2). Ein

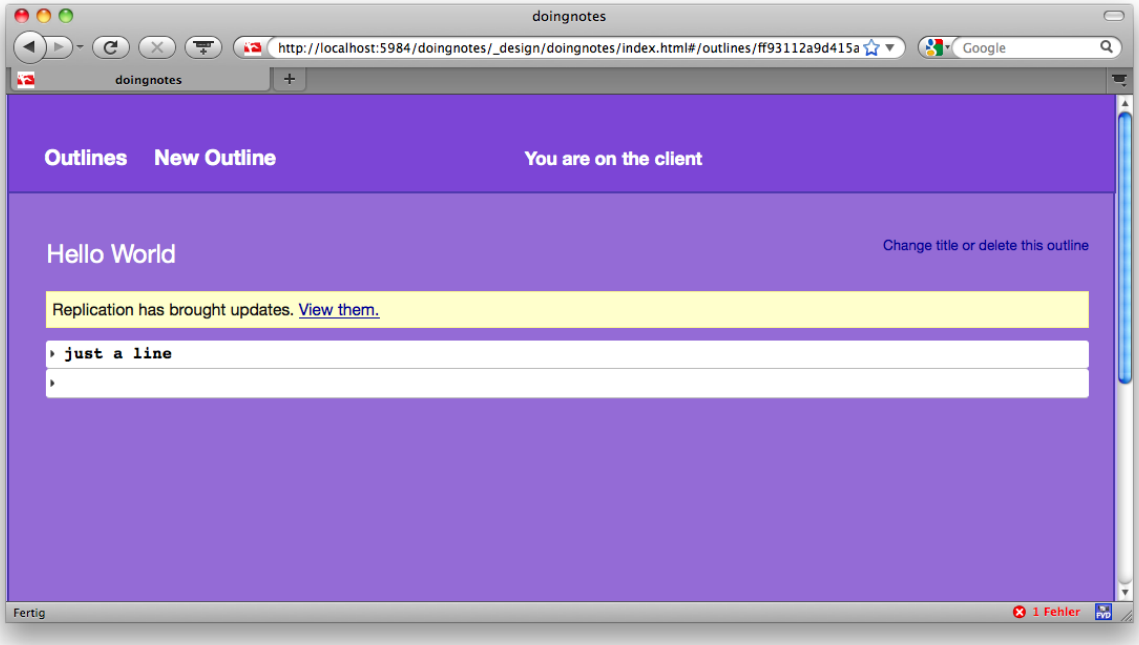

Abbildung 9.4: Screenshot: Outline mit Hinweis auf gerade stattgefundene Replikation

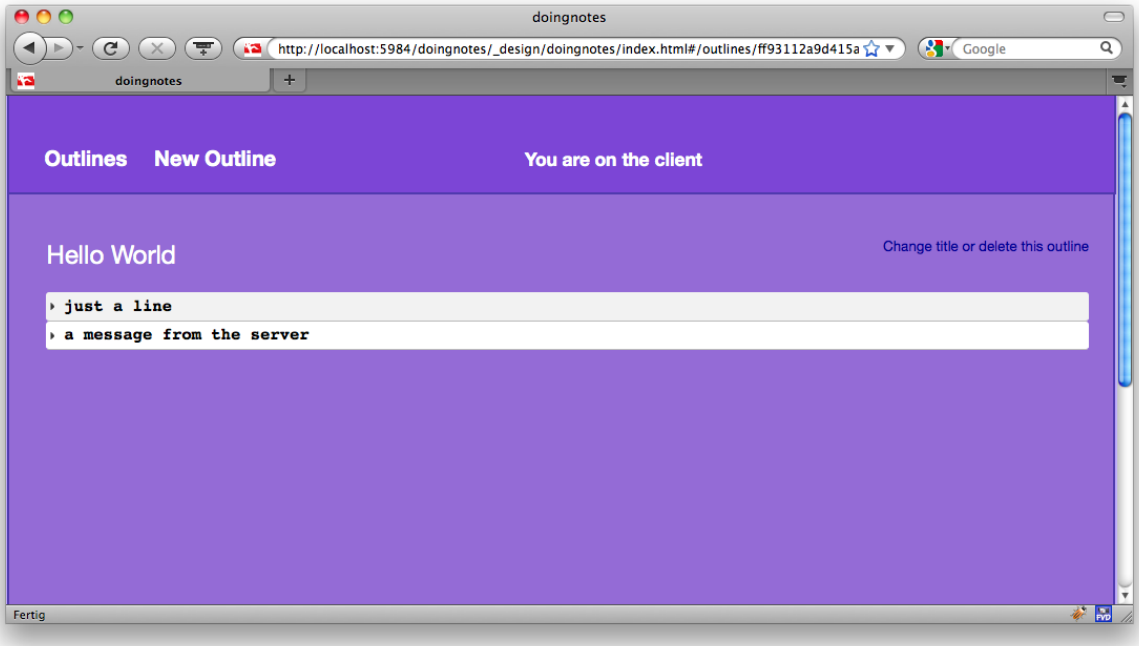

Abbildung 9.5: Screenshot: Outline nach der Aktualisierung

Append-Konflikt entsteht, wenn mehrere Benutzer an der gleichen Stelle gleichzeitig eine neue Zeile einfügen. Ein Write-Konflikt liegt vor, wenn zwischen zwei Replikationsvorgängen der Text einer Zeile von mehr als einem Benutzer gleichzeitig geändert wurde.

9.2.3.1 Erzeugung von Konflikten für Testzwecke

Ein Write-Konflikt kann bspw. erzeugt werden, wenn die folgenden Schritte in der angegebenen Reihenfolge nacheinander ausgeführt werden, wobei Client und Server vertauscht werden können:

- die Server-CouchDB-Instanz stoppen
- Text in das Client-Fenster schreiben
- die Client-Instanz stoppen
- die Server-Instanz starten
- Text in das Server-Fenster schreiben
- die Client-Instanz starten

Es ist darauf zu achten, dass die Instanzen vollständig heruntergefahren bzw. gestartet wurden, bevor der nächste Schritt ausgeführt wird. Nur dann kann die automatische Replikation lange genug ausgesetzt werden, so dass wirklich ein Konflikt entsteht.

Um einen Append-Konflikt zu erzeugen, muss in beiden Fenstern nach der gleichen Zeile eine neue Zeile eingefügt werden, anstatt Text in sie einzugeben. Das System unterstützt ebenfalls die Behandlung von Konflikten beider Arten in derselben Zeile.

9.2.3.2 Behandlung

Append-Konflikte

Entsteht durch ein Update auf dem Server ein Append-Konflikt, wird er vom System automatisch gelöst. Es wird ein Hinweis mit der Meldung „Replication has automatically solved updates.“ eingeblendet. Darüberhinaus werden die vom Konflikt betroffenen Zeilen grün eingefärbt (s. Abb. 9.6).

Der Konflikt wird so aufgelöst, dass die zum früheren Zeitpunkt erstellte Zeile vor der später erstellten eingefügt wird. Diese Sortierung ist aber nicht verlässlich, da die Uhren auf den Systemen auf denen sie erstellt wurden unter Umständen nicht gleich gehen. Bei gleichzeitiger Konfliktlösung auf zwei Rechnern (Clients) nimmt jeder Rechner die Sortierung gleich vor. Es ist also

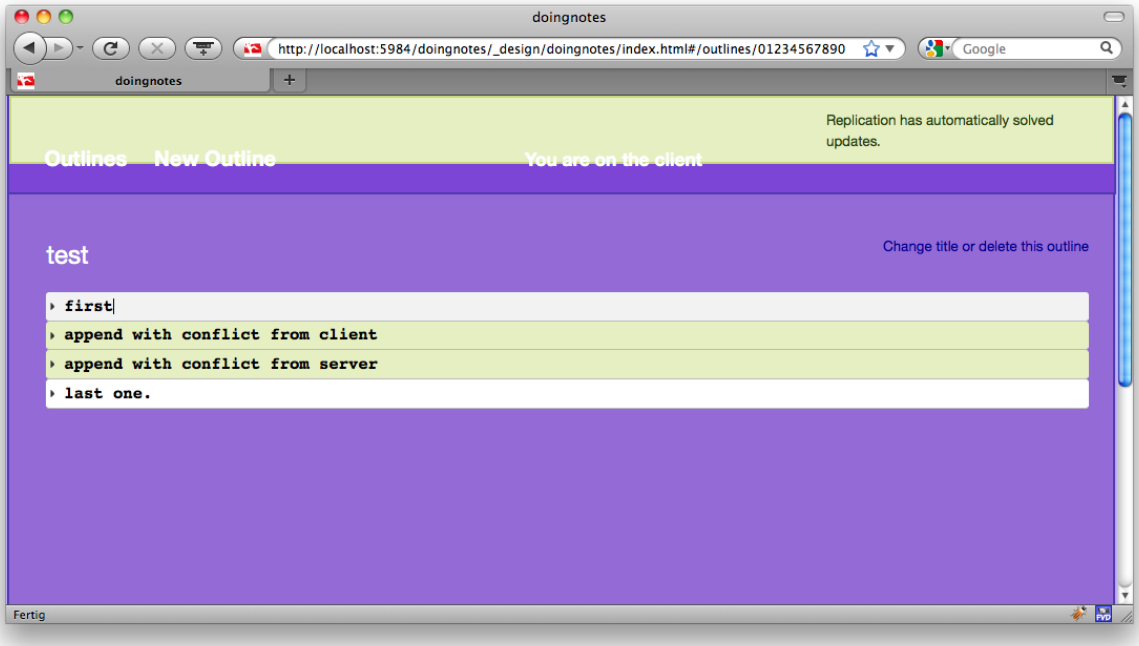

Abbildung 9.6: Screenshot: Automatisch gelöster Append-Konflikt

ausgeschlossen, dass durch die Konfliktlösung ein neuer Konflikt erscheint. Mehr Details darüber finden sich in Abschnitt 8.7.1.

Write-Konflikte

Entsteht durch ein Update ein Write-Konflikt, muss der Benutzer entscheiden, welche Version im System verbleiben und welche verworfen werden soll oder eine aggregierte Version erstellen. Es wird ein Hinweis mit der Meldung „Replication has caused one or more conflicts." eingeblendet. Darüberhinaus werden die vom Konflikt betroffenen Zeilen rot eingefärbt (s. Abb. 9.7).

Der Write-Konflikt muss manuell gelöst werden. Dazu wird der Benutzer mit einer Maske konfrontiert, in der für jede Zeile beide Versionen sichtbar sind (s. Abb. 9.8). Er kann sich jetzt für eine der Versionen entscheiden und diese vor dem Speichern nach Belieben editieren. Durch die unterschiedliche Beschriftung der Speichern-Buttons „Keep overwritten version" und „Keep winning version" wird angedeutet, welche Version in der CouchDB-internen Konfliktbehandlung als Gewinnerin hervorging. Mehr Details finden sich in Abschnitt 8.7.2.

Die Konflikte werden also zeilenweise gelöst. Nach dem Speichern der gewünschten Version einer Zeile ist diese konfliktfrei. Die Konfliktlösungs-Maske wird pro Zeile aus- und die gespeicherte Zeile eingeblendet. Wenn alle Konflikte gelöst wurden, wird dies durch eine Benachrichtigung mitgeteilt (s. Abb. 9.9).

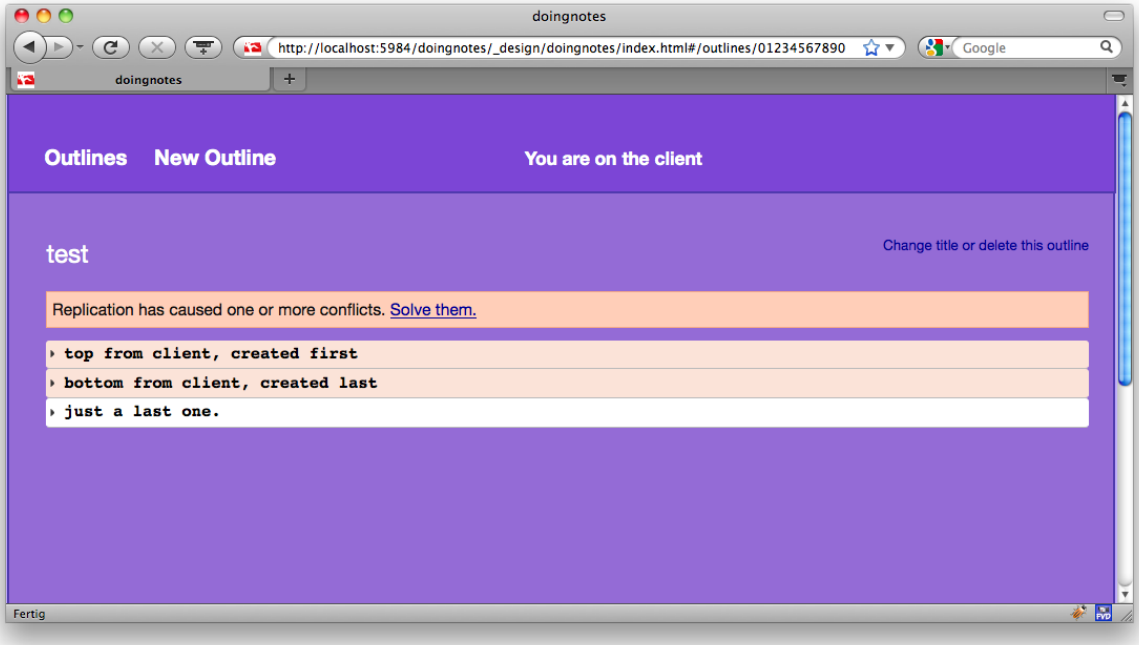

Abbildung 9.7: Screenshot: Hinweis auf einen ungelösten Write-Konflikt

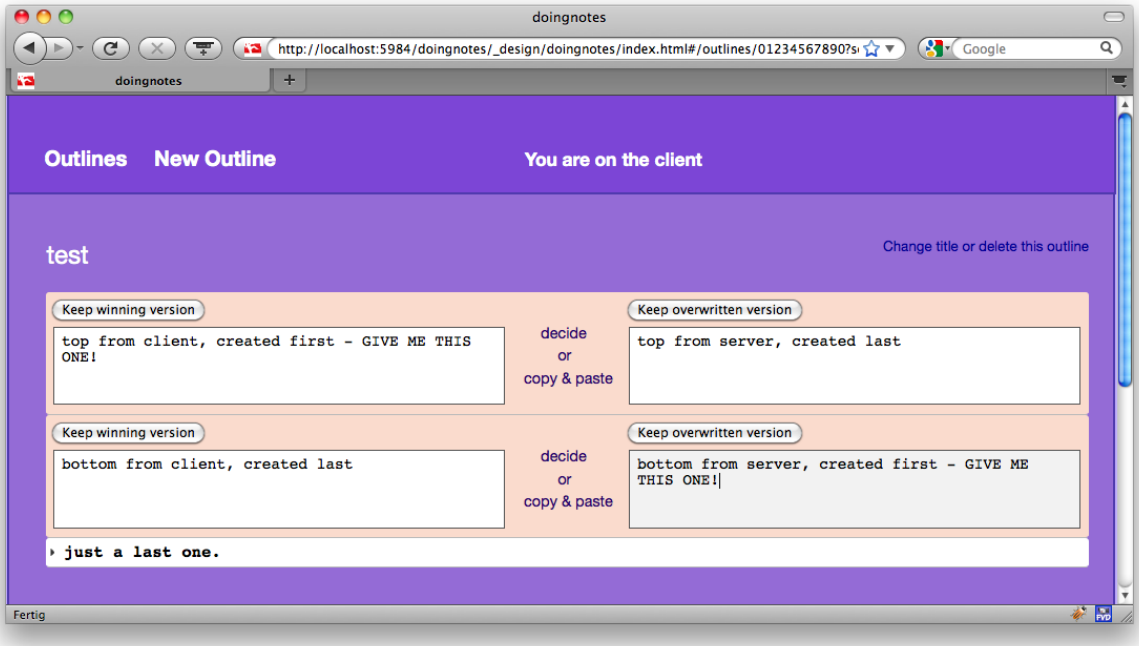

Abbildung 9.8: Screenshot: Manuelle Lösung eines Write-Konflikts

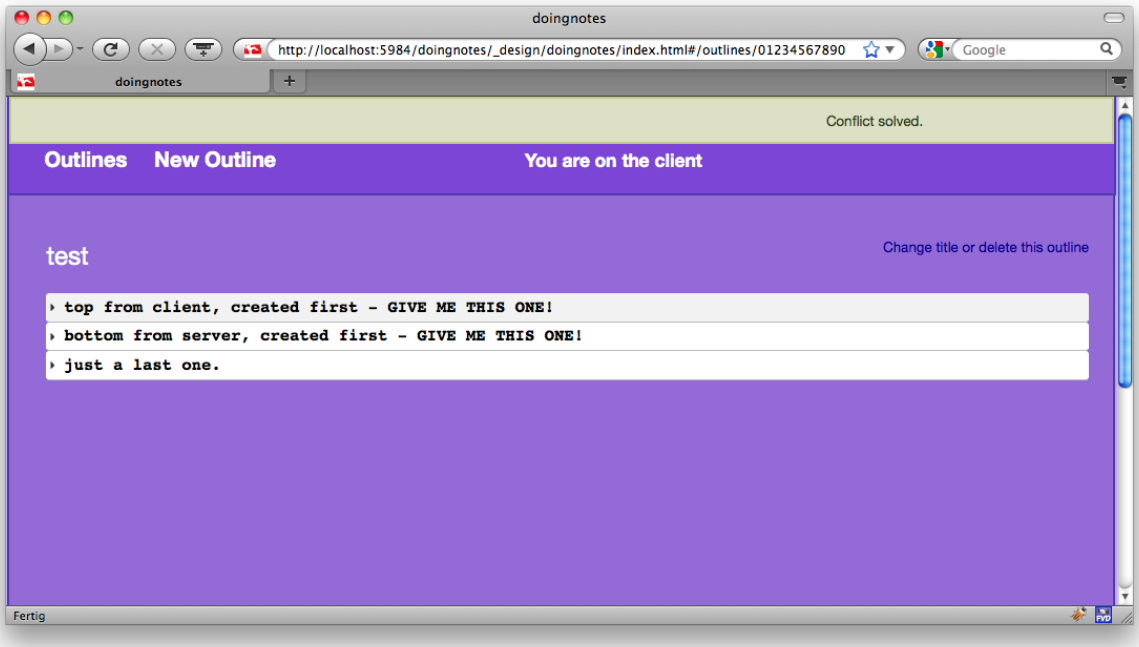

Abbildung 9.9: Screenshot: Gelöster Write-Konflikt

9.2.4 Hilfestellung zur Erzeugung der Konflikte

Zur leichteren Erzeugung der Konflikte wurden einige Makros definiert, die die Datenbank in genau definierte Zustände bringen. Ein Konflikt kann in CouchDB nicht manuell erzeugt werden. Die implementierten Makros arbeiten die im letzten Abschnitt beschriebenen Schritte programmatisch ab. Die Makros wurden als *Rake-Tasks* implementiert. Rake ist ein Build-Tool, geschrieben in Ruby. Es ähnelt dem Programm make: Kommandos werden abhängig von Bedingungen in einer Reihenfolge ausgeführt [Wei08]. Benutzer können in Ruby-Syntax eigene Abfolgen von Befehlen, sogenannte Rake-Tasks, definieren.

Um Rake nutzen zu können, wird eine Installation von Ruby [Mat10] benötigt. Außerdem müssen im Rakefile die URLs bzw. Ports der CouchDB-Instanzen angepasst werden. In der Voreinstellung befinden sich sowohl Client als auch Server auf localhost auf den Ports 5984 bzw. 5985.

In der Datei `Rakefile` im Root-Verzeichnis des Projekts finden sich folgende Rake-Tasks:

couch:recreate_host *Reset* (Löschen und Neu erstellen) der Datenbank; Anwendung wird in die Client-CouchDB-Instanz deployt

couch:recreate_server Reset der Datenbank; Anwendung wird in die Server-CouchDB-Instanz deployt

couch:wait Wartet zwei Sekunden bevor der nächste Schritt ausgeführt wird

couch:start_host Startet die Client-CouchDB-Instanz

couch:start_server Startet die Server-CouchDB-Instanz

couch:stop_host Stoppt die Client-CouchDB-Instanz

couch:stop_server Stoppt die Server-CouchDB-Instanz

couch:writeconflict Reset der Datenbank; Erzeugt ein Outline mit einem Write-Konflikt

couch:twowriteconflicts Reset der Datenbank; Erzeugt ein Outline mit zwei Write-Konflikten

couch:appendconflict Reset der Datenbank; Erzeugt ein Outline mit einem Append-Konflikt

couch:appendandwriteconflict Reset der Datenbank; Erzeugt ein Outline mit einem Append- und einem Write-Konflikt

Für die Tasks, die sich auf Client oder Server beziehen, existieren jeweils auch Schritte, die die Anweisung auf beiden Instanzen gleichzeitig ausführen.

10 Bewertung und Ausblick

Für eine Beurteilung des Erfolgs wird das entwickelte System zunächst mit den in Kapitel 6 aufgestellten Anforderungen verglichen. Anhand dessen werden das Ergebnis bewertet und bisher ungelöste Probleme aus der Implementierung diskutiert. Dann wird ein Ausblick auf Zukunftsperspektiven der verwendeten Technologien gegeben. Den Abschluss bilden Empfehlungen für die weitere Entwicklung bzw. Forschung.

10.1 Bewertung des Ergebnisses

Die insgesamt siebzehn aufgestellten Muss-Kriterien (s. Abschnitt 6.1.1) konnten vollständig umgesetzt werden. Dagegen konnten von sechzehn Kann-Kriterien (s. Abschnitt 6.1.2) nur vier implementiert werden. Aus Zeitgründen nicht umgesetzt wurden Verschieben, Löschen, Kommentierung und Versionierung von Zeilen im Gliederungseditor, gezieltes Replizieren von einzelnen Outlines und explizites Ein- und Ausstellen der Replikation.

Für den produktiven Einsatz müsste vor allem die Konfliktbehandlung weiterentwickelt werden. Wie in Abschnitt 7.4.3 beschrieben führen Konflikte, die durch gleichzeitiges abweichendes Ein- oder Ausrücken von Zeilen entstehen, zu Fehlern beim Aufbau der Outlines. Werden Zeilen gleichzeitig in drei oder mehr Repliken verändert, führt dies ebenfalls zu Konflikten, die nicht korrekt abgefangen werden. Dies ist in erster Linie ein Problem der Darstellung: die Benutzeroberfläche für das manuelle Lösen von Konflikten ist, wie in Abschnitt 8.7.2 dargestellt wurde, nur für zwei unterschiedliche Versionen einer Zeile konzipiert. Auch wenn dies im produktiven Einsatz nicht häufig vorkäme, müsste die manuelle Zusammenführung einer höheren Anzahl von Zeilenversionen im Hinblick auf die Gestaltung der Benutzeroberfläche grundlegend anders angegangen werden.

Den nichtfunktionalen Anforderungen wurde weitestgehend entsprochen. Verbesserungswürdig ist der Quelltext in Bezug auf Einfachheit und Redundanzfreiheit: Um ein durchgehend nach der MVC-Architektur aufgebautes Programm zu erhalten, müssten die Funktionen zur Steuerung der Replikation in den objektorientierten Programmierstil umgewandelt werden. Dies wurde bereits bei der Konflikterkennung, -präsentierung und -bearbeitung umgesetzt. Den Anforderungen an die Benutzeroberfläche (vgl. Abschnitt 6.2.3) kommt die Anwendung nach. Jedoch reagiert die Benutzeroberfläche auf Benutzereingaben aufgrund des notwendigen Verbindungsaufbaus zwischen Frontend und Datenbank mitunter so verzögert, dass der Arbeitsfluss für wenige Sekunden

unterbrochen wird. Es bleibt zu prüfen, ob dies mit einer Optimierung der Benutzeroberfläche behoben werden kann.

Trotz der beschriebenen Mängel konnte die in der Aufgabenstellung beschriebene Anwendung weitestgehend erfolgreich entworfen und umgesetzt werden. Die wichtigste Leistung der Arbeit besteht darin, dass mit dem entwickelten Programm ein bestimmtes Paradigma der Internetbenutzung verwirklicht wird, nämlich das der Peer-to-Peer-Kommunikation. Im Vergleich zu üblichen Client-Server-Anwendungen haben Benutzer bei der hier konzipierten Architektur mehr Kontrolle über ihre Daten. Gemeinsames Arbeiten kann umgesetzt werden, ohne auf eine kontinuierliche Netzwerkverbindung und ständig verfügbare Server angewiesen zu sein.

10.2 Die Zukunft der eingesetzten Technologien

Die Implementierung gestaltete sich insgesamt zeitaufwändiger als gedacht. Um die HTTP-API, Replikation, Konfliktbehandlung und Überwachung der Datenbank auf Änderungen passend anzuwenden, musste eine beträchtliche Grundlagenarbeit geleistet werden. Für die Umsetzung der eigentlichen Fachlogik verblieb vergleichsweise wenig Zeit. Jedoch werden die eingesetzten Technologien fortlaufend verbessert und neue Bibliotheken und Frameworks entwickelt, durch die der Arbeitsaufwand für ähnliche Projekte in Zukunft geringer ausfallen dürfte.

So enthält die CouchDB-Version 1.0 einige Features, die die Verbesserung der umgesetzten Anwendung deutlich vereinfachen kann [Leh10a]. Ihre Veröffentlichung wird auf das Ende der Bearbeitungszeit dieser Arbeit fallen [Sla10]. Mit CouchDB in der Version 1.0 wird es beispielsweise möglich sein, Dokumente einzeln unter Angabe ihrer ID zu replizieren, und nicht mehr nur die Datenbank als Ganzes. So kann eine gezielte Replikation von Outlines einfacher erfolgen. Des Weiteren wird die Unterstützung für das Betriebssystem Windows verbessert, wodurch die Plattformunabhängigkeit der Anwendung wachsen wird. Für zukünftige Releases von CouchDB ist außerdem geplant, Sharding nativ zu unterstützen [Lehnardt, Jan, persönliche Mitteilung, 09.07.2010]. Damit kann das Aufsetzen von CouchDB-Lounge in Zukunft entfallen.

Unter den aktuellen Neuentwicklungen ist das Framework *Evently* zu erwähnen [And10]. Damit kann ähnlich wie mit Sammy das Routing einer Anwendung umgesetzt werden, es wurde jedoch explizit für eventbasierte Anwendungen mit CouchDB entwickelt. Evently stellt eine Verbindung zwischen CouchDB-Views, dem Changes-Feed, HTML-Templates und definierten JavaScript-Callbacks her und gibt eine Struktur für die Organisierung des Quelltexts vor. Gegenüber den in dieser Arbeit verwendeten Mitteln kann Evently deutliche Produktivitätsvorteile bieten.

10.3 Empfehlungen für die Weiterentwicklung

Die Entwicklung des Gliederungseditors kann ohne größere Hindernisse fortgesetzt werden. Durch die Implementierung der Baumstruktur können Löschen bzw. Verschieben einer Zeile einfach umgesetzt werden, indem die Zeile nicht mehr angezeigt bzw. innerhalb des Baums an ihrer neuen Position eingebaut wird. Spalten im Gliederungseditor können ebenfalls ohne Schwierigkeiten umgesetzt werden, indem den Zeilen mehrere Textareas zugewiesen werden.

Ein weiteres Arbeitsfeld wäre die Umsetzung von Zugriffskontrolle und Benutzerverwaltung. Einzelne Outlines könnten vom Benutzer als öffentlich oder privat gekennzeichnet werden und dementsprechend für die Replikation freigegeben sein oder nicht. Anwendungen, die mit verteilten Daten auf mobilen Endgeräten arbeiten, stellen erhöhte Anforderungen an Sicherheitsvorkehrungen:

> Providing high availability and the ability to share data despite the weak connectivity of mobile computing raises the problem of trusting replicated data servers that may be corrupt. This is because servers must be run on portable computers, and these machines are less secure and thus less trustworthy than those traditionally used to run servers. [...] Portable machines are often left unattended in unsecured or poorly secured places, allowing attackers with physical access to modify the data and programs on such computers. [Spr97, Kap. 1]

Demzufolge wäre für einen Produktiveinsatz der Implementierung von Zugriffskontrolle ein hoher Stellenwert einzuräumen. Die Priorität für die Weiterentwicklung sollte nach Meinung der Autorin allerdings zuerst darin liegen, die Peer-to-Peer-Fähigkeiten der Anwendung weiter auszubauen. Instanzen der Anwendung könnten freigegebene Outlines über einen Webservice propagieren. Protokolle wie *Bonjour* ermöglichen eine automatische Erkennung von Netzwerkdiensten in lokalen IP-Netzen [App10]. Ein solches Protokoll könnte verwendet werden, damit Instanzen der Anwendung sich gegenseitig in einem Netzwerk erkennen und die Möglichkeit bieten können, Outlines direkt miteinander zu replizieren. Damit könnten Dokumente zB. in einem Büro oder auf einer Konferenz auch ohne Internetverbindung gleichzeitig bearbeitet werden.

Die Umsetzung der Aufgabenstellung kann als gelungen bezeichnet werden. Es wird allerdings noch beträchtlichen Entwicklungsaufwands bedürfen, bis die entstandene Anwendung für alle in Abschnitt 3.1.2 aufgezählten Einsatzmöglichkeiten produktiv eingesetzt werden kann. Wenn mehrere Benutzer ein Outline zeitgleich in größerem Umfang bearbeiten und erst nach zahlreichen Änderungen synchronisieren, werden die Anzahl und Komplexität der Konflikte von der Anwendung in der aktuellen Version noch nicht befriedigend und stabil bewältigt. Mit CouchDB und den anderen evaluierten Technologien kann ein Verteiltes System gut umgesetzt werden, das Mergen der Daten liegt jedoch im Aufgabenbereich der Anwendungsentwicklerin. Das CouchDB Entwicklerteam plant jedoch, vorgefertigte Lösungen für häufig vorkommende Konflikt-Szenarien

anzubieten [Lehnardt, Jan, persönliche Mitteilung, 09.07.2010]. Nichtsdestoweniger ist das Mergen beim vorliegenden Anwendungsfall eine überdurchschnittliche Herausforderung, da durch den gewählten Lösungsansatz die Dokumente (die einzelnen Zeilen einer Outline) sehr granular gewählt und stark verknüpft sind. Einsatzgebiete, in denen seltener gleichzeitig an einem Dokument gearbeitet wird und Konflikte daher seltener auftreten, dürften mit deutlich weniger komplexen Lösungen auskommen. Denkbar sind Adressbücher, Kalender, Kundendaten, oder auch Dienste, in denen Nachrichten ausgetauscht werden. In [And10, Kap. 10] und [Leh10a] finden sich weitere Anregungen.

Anhang

A.1 Abkürzungen

Hier sind alle Abkürzungen aufgeführt, sofern sie nicht als allgemein bekannt vorausgesetzt werden können oder im Text nur in Verbindung mit ihrer Ausschreibung auftreten.

Abkürzung	Beschreibung
AJAX	Asynchronous JavaScript And XML
AMI	Amazon Machine Image
API	Application Programming Interface
AWS	Amazon Web Services
BDD	Behaviour Driven Development
CAP	Concurrency / Availability / Partition Tolerance
CGI	Common Gateway Interface
CSS	Cascading Stylesheets
DOM	Document Object Model
EBS	Elastic Block Storage
EC2	JavaScript Object Notation
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MVCC	Multi Version Concurrency Control
PaaS	Platform as a Service
RDBMS	Relational Database Management System
REST	Representational State Tansfer
S3	Simple Storage Service
SaaS	Software as a Service
TDD	Test Driven Development
UUID	Universally Unique Identifier

Abbildung A.1: Abkürzungsverzeichnis

A.2 Ergänzungen zur Analyse

A.2.1 Omni-Outliner

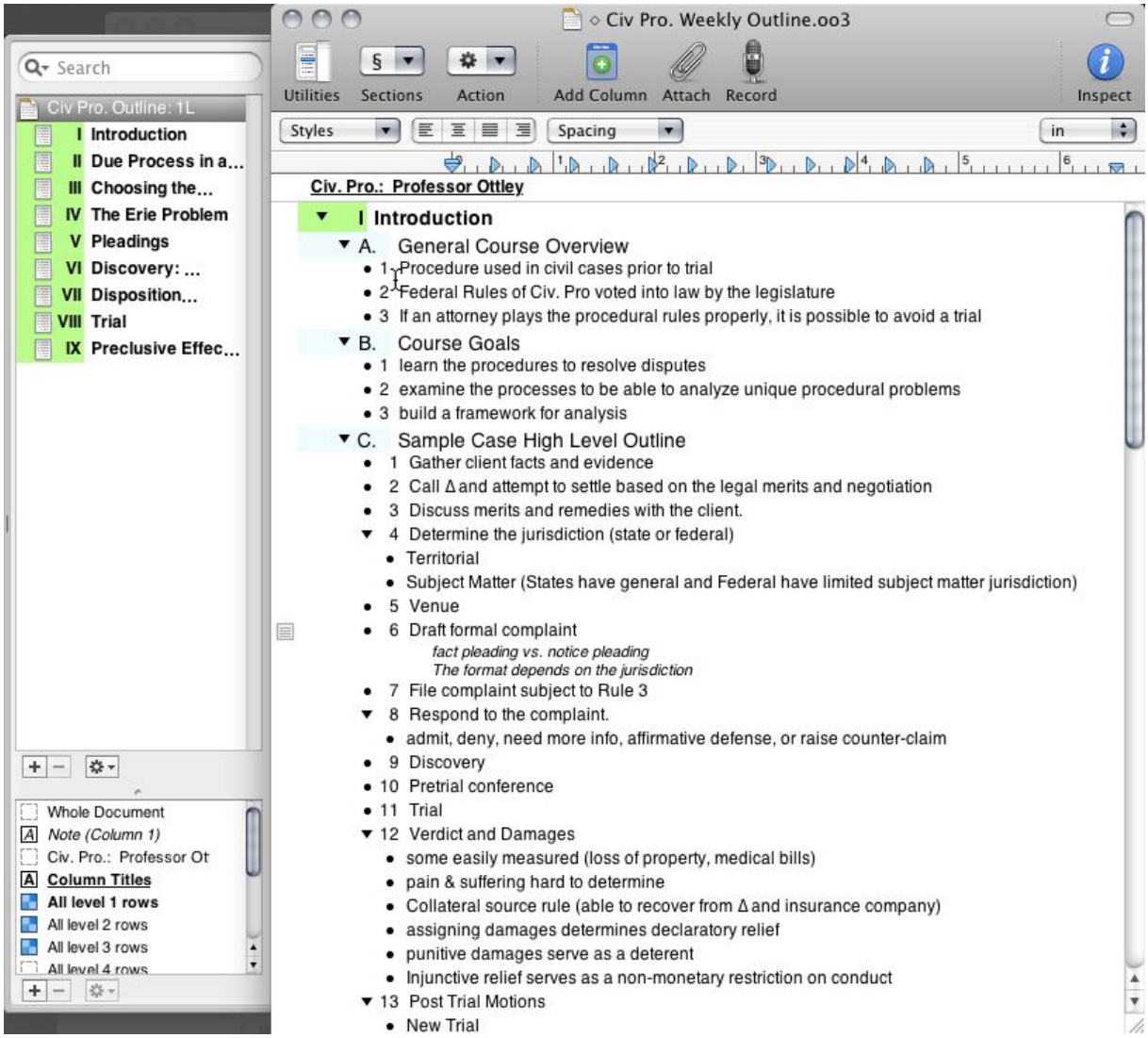

Abbildung A.2: Screenshot von OmniOutliner [Law10]

A.2.2 Der Hype-Zyklus von Gartner

Aus dem Hype-Zyklus sind die Phasen der öffentlichen Aufmerksamkeit ablesbar, die eine neue Technologie nach ihrer Einführung durchläuft. Die X-Achse bezeichnet die Zeit nach der Einführung, die Y-Achse die Aufmerksamkeit für die Technologie.

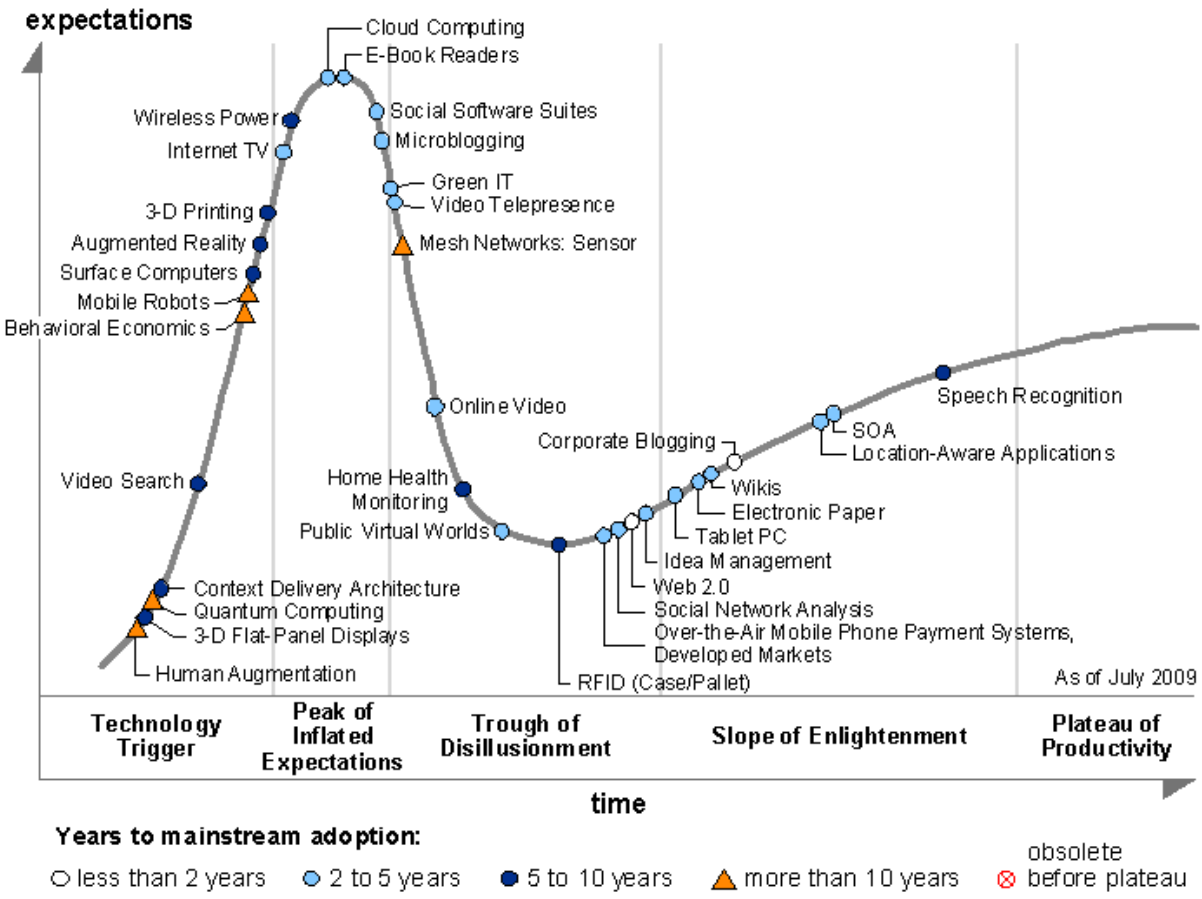

Abbildung A.3: Hype-Zyklus von Gartner 2009, [Gar09]

A.3 Ergänzungen zum Technischen Hintergrund

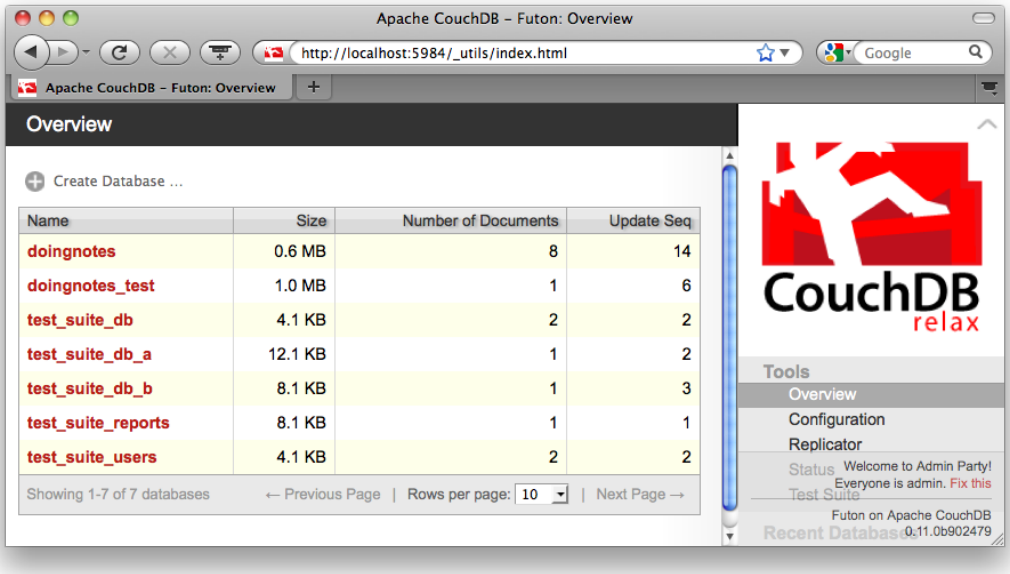

Abbildung A.4: CouchDB Futon: Overview

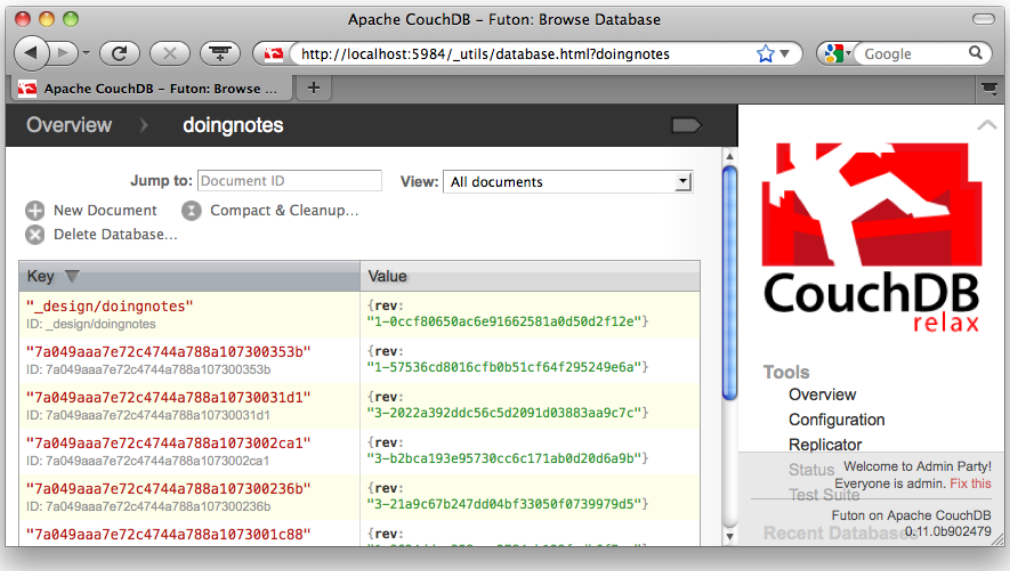

Abbildung A.5: CouchDB Futon: Browse Database

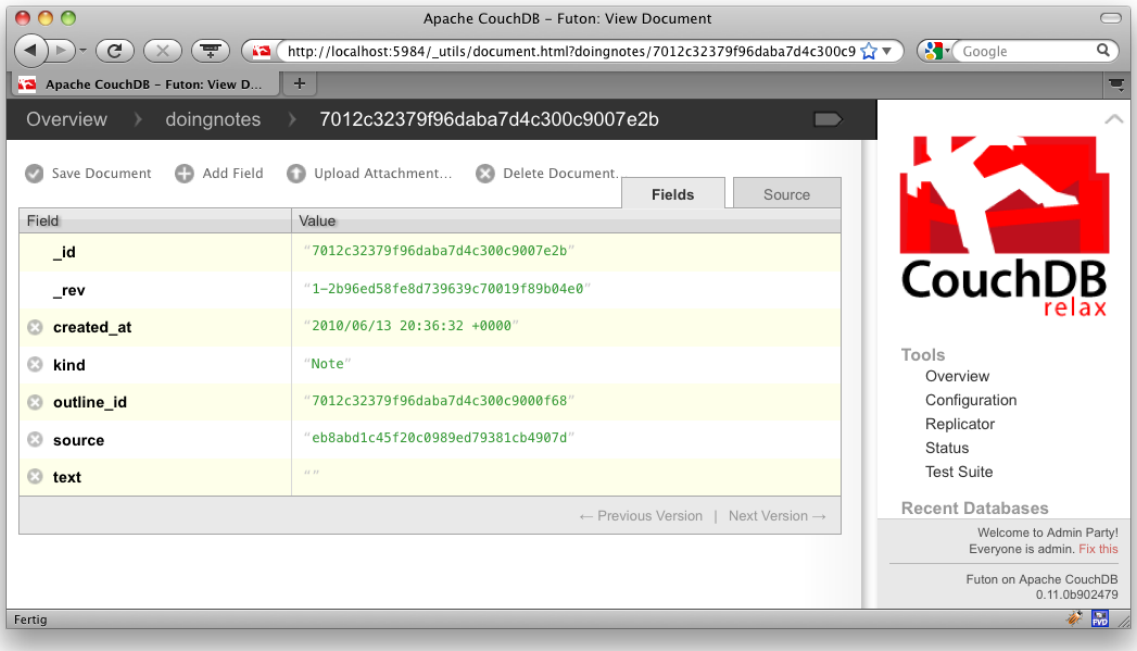

Abbildung A.6: CouchDB Futon: Document

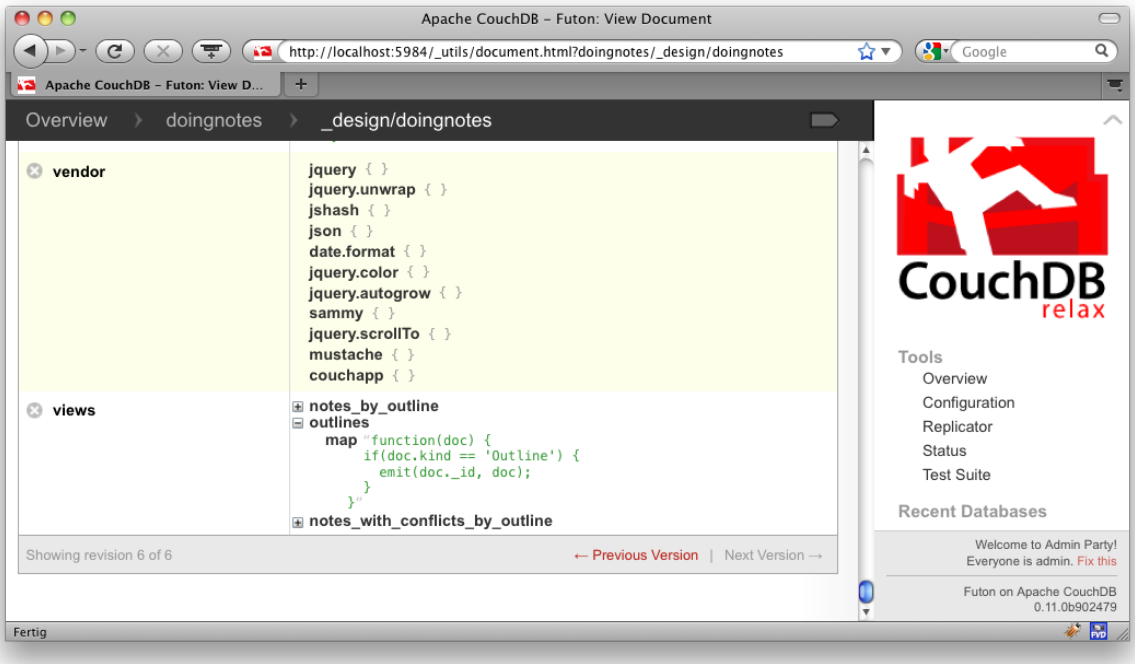

Abbildung A.7: CouchDB Futon: Design Document

A.4 Ergänzungen zu den Systemanforderungen

A.4.1 Funktionale Anforderungen

In Tabelle A.8 finden sich die funktionalen Anforderungen in tabellarischer Form, die in Abschnitt 6.1 textuell ausgeführt sind. Die Tabelle ist nach Bereichen und absteigend nach Priorität geordnet. Für die Priorität werden die Ziffern *1* (höchste Priorität), *2* (mittlere Priorität) und *3* (niedrigste Priorität) verwendet.

Bereich	Nr.	Beschreibung	Priorität
Outlines	**FA100**	Erstellen	1
	FA101	Titel ändern	1
	FA102	Löschen	2
Zeilen	**FA200**	Hinzufügen	1
	FA201	Navigieren mit Funktionstasten	1
	FA202	Inhalt bearbeiten	1
	FA203	Bei Verlassen automatisch speichern	1
	FA204	Ein- und ausrücken	1
	FA205	Blockweise ein- und ausrücken	1
	FA206	Warnung vor Datenverlust bei Schließen des Fensters	2
	FA207	Nach oben/unten verschieben	2
	FA208	Größe anpassen an Länge des Textes	2
	FA209	Ein- und ausklappen	2
	FA210	Einklappstatus lokal speichern	2
	FA211	Revisionen speichern	3
	FA212	Zwischen Revisionen wechseln	3
	FA213	Kommentieren	3
	FA214	Löschen	3
Replikation	**FA300**	Outlines zum Server replizieren	1
	FA301	Änderungen an Outlines zum Server replizieren	1
	FA302	Outlines von Anderen erhalten	1
	FA303	Änderungen an Outlines von Anderen erhalten	1
	FA304	Benachrichtigung bei Änderungen von Anderen	1
	FA305	Wiederaufnahme oder Hinweis bei Internetzugang	2
	FA306	Outlines veröffentlichen und auswählen	3
	FA307	Statusmeldung über Verbindungszustand	3
	FA308	An- und ausstellen	3
Konflikte	**FA400**	Eine Konfliktart automatisch lösen	1
	FA401	Eine Konfliktart manuell lösen	1
	FA402	Kombination aus unterschiedlichen Konfliktarten lösen	2
	FA403	Konflikte zwischen mehr als zwei Repliken lösen	2
	FA404	Mehrere Konfliktarten automatisch lösen	3
	FA405	Mehrere Konfliktarten manuell lösen	3

Abbildung A.8: Anforderungen an das System

A.5 Ergänzungen zur Systemdokumentation

A.5.1 Fachklassendiagramm

In Abbildung A.9 ist eine Übersicht über die zentralen Fachklassen der Anwendung zu finden. Da JavaScript hier nicht als klassenorientierte Sprache verwendet wird, handelt es sich nicht um Klassen im engeren Sinn. Eine UML-Klasse wird hier für eine JavaScript-Funktion verwendet. Als Attribute werden lokale Variablen verwendet. Um Klassenmethoden umzusetzen, wurden die entsprechenden Funktionen dem Prototypen der Funktion hinzugefügt. Eine UML-Assoziation bedeutet, dass die „Klasse“, die eigentlich eine Funktion ist, eine andere „Klasse“ in einer ihrer Methoden aufruft.

Durch das Diagramm wird ausgedrückt, wie sich die Fachklassen zueinander Verhalten. Die Controller ganz links greifen auf die Mustache-Views zu, die wiederum ihre Daten aus den Models beziehen. Der `ConflictDetector` wird unabhängig von den Fachklassen durch Input aus der Datenbank aufgerufen. Er ruft ggf. den `ConflictPresenter` auf, um Konflikte darzustellen, oder den `ConflictResolver`, um sie aufzulösen. Dafür benötigt letzterer auch Daten aus einer `NoteView` und der `NotesCollection`. Der `ConflictResolver` kann auch von einer `NotesView` aufgerufen werden, um einen Write-Konflikt darzustellen.

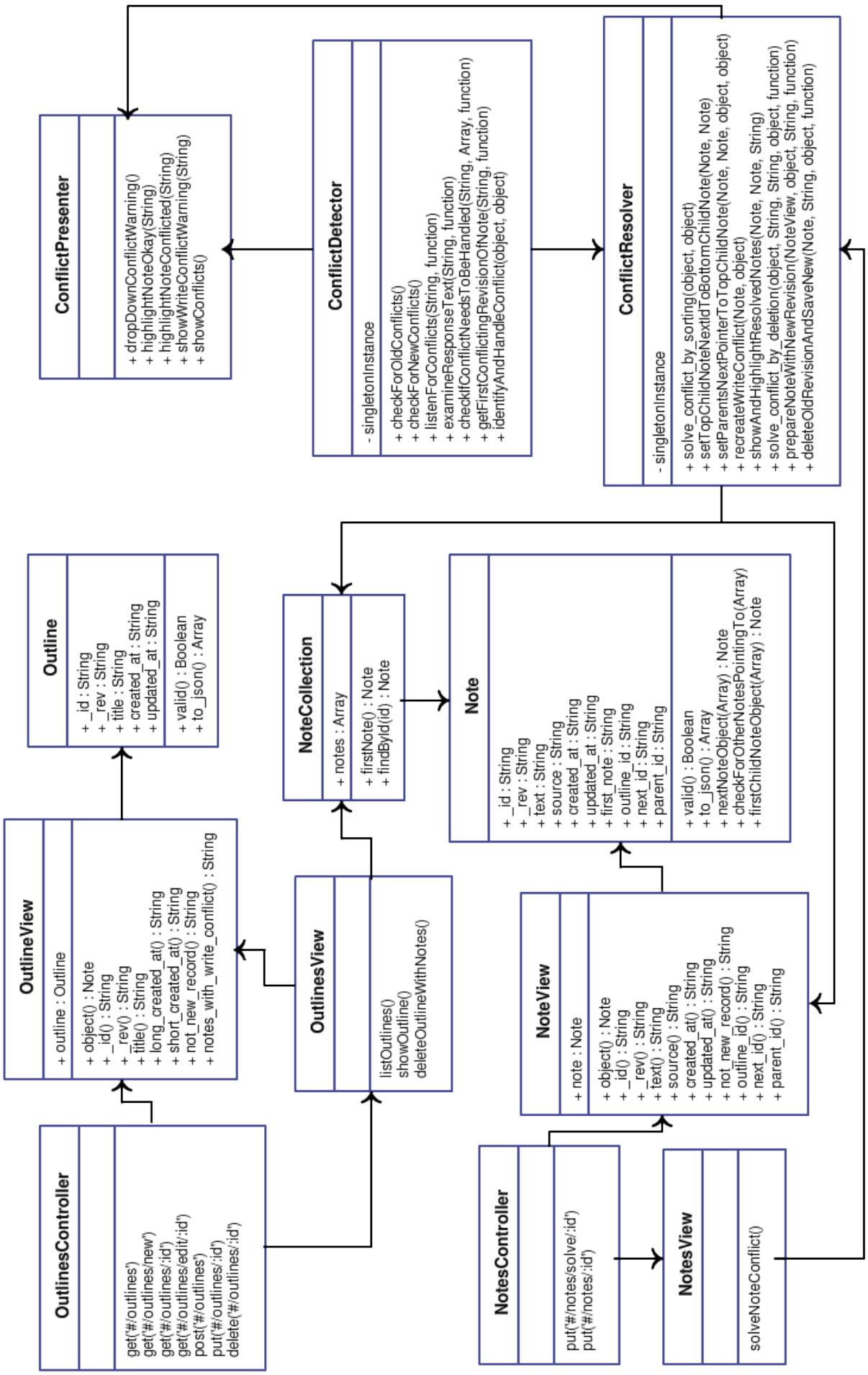

Abbildung A.9: Fachklassendiagramm

A.5.2 Abstraktion der grundlegenden Datenbankoperationen

```
var Resources = function(app, couchapp) {
  this.helpers({
    new_object: function(kind, callback) {
      this.partial(template_file_for(kind, 'new'), callback);
    },
    \\ [...]

    load_object_view: function(kind, id, callback){
      var context = this;
      couchapp.db.openDoc(id, {
        success: function(doc) {
          var _prototype = eval(kind);
          var view_prototype = eval(kind + 'View');
          var view = new view_prototype(new _prototype(doc));
          if(doc) {
            callback(view);
          } else {
            context.flash = {message: kind + ' with ID "' + id + '" not found.', type: '
                error'};
          }
        },
        error: function() {
          context.notFound();
        }
      });
    },

    \\ [...]
  });
};
```

Listing A.1: Auszug aus `/_attachments/app/lib/resources.js`

A.5.3 Einrückung von Zeilen im Outline

```
<li class="edit-note" id="edit_note_2">
  <form class="edit-note" action="#/notes/2" method="put" accept-charset="utf-8">
    <span class="space"> </span>
    <a class="image"> </a>
    <textarea class="expanding" id="edit_text_2" name="text">Some text</textarea>
    <input type="submit" value="Save" style="display:none;"/>
  </form>

```

```
  <ul class="indent">
    <li class="edit-note" id="edit_note_2a">
      <form class="edit-note" action="#/notes/2a" method="put" accept-charset="utf-8">
        <span class="space"> </span>
        <a class="image"> </a>
        <textarea class="expanding" id="edit_text_2a" name="text">More text</textarea>
        <input type="submit" value="Save" style="display:none;"/>
      </form>
    </li>
  </ul>
</li>
```

Listing A.2: Zeile mit einem Kindknoten

A.5.4 Rendern der Zeilen eines Outlines

```
renderNotes: function(context, notes, counter){
  if (notes.notes.length == 0) return;
  if (notes.notes.length == 1) {
    context.unbindSubmitOnBlurAndAutogrow();
    context.bindSubmitOnBlurAndAutogrow();
    $('#spinner').hide();
    context.i = 0;
  }
  if(typeof(context.i)=="undefined"){
    context.i = counter;
  } else {
    context.i = context.i-1;
  }
  var note_object = notes.findById(this.id());
  var child_object = note_object.firstChildNoteObject(notes.notes);
  var next_object = note_object.nextNoteObject(notes.notes);

  notes.notes = notes.notes.remove(note_object);

  if(typeof(child_object)!="undefined"){
    this.renderFollowingNote(context, child_object, function(child){
      child.renderNotes(context, notes);
    });
  }
  if(typeof(next_object)!="undefined"){
    this.renderFollowingNote(context, next_object, function(next){
      next.renderNotes(context, notes);
    });
  }
  if(typeof(next_object)=="undefined" && typeof(child_object)=="undefined"){
```

```
    context.unbindSubmitOnBlurAndAutogrow();
    context.bindSubmitOnBlurAndAutogrow();
  }
},

renderFollowingNote: function(context, note_object, callback){
  var this_note = this;
  context.partial('app/templates/notes/edit.mustache', {_id: note_object._id, text:
      note_object.text}, function(html) {
    if(typeof note_object.parent_id != "undefined"){
      $(html).appendTo(this_note.note_target.closest('li')).wrap('<ul class="indent"></
          ul>');
      callback(this_note.firstChildNote());
    } else {
      $(html).insertAfter(this_note.note_target.closest('li'));
      callback(this_note.nextNote());
    }
  });
}
```

Listing A.3: Auszug aus `/_attachments/app/helpers/note_element.js`

A.5.5 Überwachen der Datenbank auf Änderungen von Anderen

```
checkForUpdates: function(couchapp){
  var context   = this;
  var source    = context.getLocationHash();
  var url       = config.HOST + '/' + config.DB +
                  '/_changes?filter=doingnotes/changed&source=' + source;

  if(context.getOutlineId()){
    $.getJSON(url, function(json){
      var since = json.last_seq;
      var xmlhttp = new XMLHttpRequest();
      xmlhttp.onreadystatechange=function() {
        if(xmlhttp.readyState == 3){
          if(xmlhttp.responseText.match(/changes/)){
            var lines = xmlhttp.responseText.split("\n");
            if(lines[lines.length-2].length != 0){
              lines = lines.remove("");
              $.each(lines, function(i, line){
                context.parseLineAndShowChangesWarning(context, couchapp, line, lines);
              });
            }
          }
          if(xmlhttp.responseText.match(/last_seq/)){
```

```
23            Sammy.log('Timeout in checkForUpdates:', xmlhttp.responseText)
24          }
25        }
26      }
27      xmlhttp.open("GET", url+ '&feed=continuous&heartbeat=5000&since=' +since, true);
28      xmlhttp.send(null);
29    });
30  }
31 }
```

Listing A.4: `/_attachments/app/helpers/replication_helpers.js`

A.5.6 Patch für den CouchDB Changes-Filter

Die Funktion `make_filter_fun` erzeugt eine Funktion, die für jede Zeile im Changes-Feed das entsprechende Dokument lädt. Die angegebene Filterfunktion entscheidet anhand der JSON-Repräsentation des Dokuments, ob die Zeile in der Rückgabe des Changes-Feeds enthalten sein soll. Das `_conflicts`-Array wurde in Zeile 17 dem JSON-Objekt hinzugefügt, so dass auch auf dieses von einer Filterfunktion berücksichtigt werden kann.

```
1  make_filter_fun(Req, Db) ->
2      Filter = couch_httpd:qs_value(Req, "filter", ""),
3      case [list_to_binary(couch_httpd:unquote(Part))
4              || Part <- string:tokens(Filter, "/")] of
5      [] ->
6          fun(DocInfos) ->
7              [{[{rev, couch_doc:rev_to_str(Rev)}]} ||
8                  #doc_info{revs=[#rev_info{rev=Rev}|_]} <- DocInfos]
9          end;
10     [DName, FName] ->
11         DesignId = <<"_design/", DName/binary>>,
12         DDoc = couch_httpd_db:couch_doc_open(Db, DesignId, nil, []),
13         #doc{body={Props}} = DDoc,
14         couch_util:get_nested_json_value({Props}, [<<"filters">>, FName]),
15         fun(DocInfos) ->
16             Docs = [Doc || {ok, Doc} <- [
17                 {ok, Doc} = couch_db:open_doc(Db, DInfo, [deleted, conflicts])
18                 || DInfo <- DocInfos]],
19             {ok, Passes} = couch_query_servers:filter_docs(Req, Db, DDoc, FName, Docs),
20             [{[{rev, couch_doc:rev_to_str(Rev)}]}
21                 || #doc_info{revs=[#rev_info{rev=Rev}|_]} <- DocInfos,
22                 Pass <- Passes, Pass == true]
23         end;
24     _Else ->
25         throw({bad_request,
```

```
            "filter parameter must be of the form `designname/filtername`"})
    end.
```

Listing A.5: Die Funktion `make_filter_fun`

Für den Test wurde ein Designdokument in die Testdatenbank eingefügt, das eine List-Funktion enthält, die alle Dokumente mit Konflikten ausgibt.

```
var ddoc = {
  _id : "_design/changes_filter",
  "filters" : {
    "conflicted" : "function(doc, req) { return (doc._conflicts);}",
  }
}

var id = db.save({'food' : 'pizza'}).id;
db.bulkSave([{_id: id, 'food' : 'pasta'}], {all_or_nothing:true});

req = CouchDB.request("GET", "/test_suite_db/_changes?filter=changes_filter/conflicted
    ");
resp = JSON.parse(req.responseText);
T(resp.results.length == 1);
```

Listing A.6: Test für die Funktion in Listing A.6

A.5.7 HTML-Datei zum Ausführen der Unit Tests

```
<html>
  <head>
    <link rel="stylesheet" href="jspec/jspec.css" type="text/css"/>
    <script src="../../vendor/jquery/_attachments/jquery.js"></script>
    <script src="jspec/jspec.js"></script>
    <script src="jspec/jspec.jquery.js"></script>
    <script src="jspec/jspec.xhr.js"></script>
    <script src="../app/lib/lib.js"></script>
    <script src="../app/lib/resources.js"></script>
    <script src="../config/config.js"></script>
    <script src="../app/helpers/key_events.js"></script>
    <script src="../app/helpers/note_element.js"></script>
    <script src="../app/helpers/outline_helpers.js"></script>
    <script src="../app/models/note.js"></script>
    <script src="../app/models/outline.js"></script>
    <script src="../app/models/note_collection.js"></script>
    <script>
      function runSuites() {
```

```
        JSpec
        .exec('note_element_spec.js')
        .exec('inserting_note_element_spec.js')
        .exec('indenting_note_element_spec.js')
        .exec('unindenting_note_element_spec.js')
        .exec('focusing_note_element_spec.js')
        .exec('rendering_note_element_spec.js')
        .exec('outline_helpers_spec.js')
        .exec('outline_spec.js')
        .exec('note_spec.js')
        .exec('note_collection_spec.js')
        .exec('resources_spec.js')
        .exec('lib_spec.js')
        .exec('conflict_spec.js')
        .run({failuresOnly: true, fixturePath: 'fixtures'})
        .report();
      }
    </script>
  </head>
  <body class="jspec" onLoad="runSuites();">
    <div id="jspec"></div>
  </body>
</html>
```

Listing A.7: `/_attachments/spec/index.html`

A.5.8 Testsuite für die CouchDB JavaScript HTTP API

A.5.8.1 Auszug aus der CouchDB JavaScript HTTP API

```
(function($) {
  \\ [...]
  $.extend($.couch, {
    \\ [...]
    db: function(name) {
      \\ [...]
      return {
        \\ [...]
        removeDoc: function(doc, options) {
          ajax({
              type: "DELETE",
              url: this.uri +
                   encodeDocId(doc._id) +
                   encodeOptions({rev: doc._rev})
            },
```

```
16            options,
17            "The document could not be deleted"
18          );
19        }
20        \\ [...]
21      };
22    }
23    \\ [...]
24  });
25 })(jQuery);
```

Listing A.8: Auszug aus `/share/www/script/jquery.couch.js`

A.5.8.2 Der Test für Listing A.8

```
1  describe 'jQuery couchdb db'
2    \\ [...]
3
4    before_each
5      db = $.couch.db('spec_db');
6      db.create();
7    end
8
9    after_each
10     db.drop();
11   end
12
13   describe 'removeDoc'
14     before_each
15       doc = {"Name" : "Louanne Katraine", "Callsign" : "Kat", "_id" : "345"};
16       saved_doc = {};
17       db.saveDoc(doc, {
18         success: function(resp){
19           saved_doc = resp;
20         },
21         error: function(status, error, reason){errorCallback(status, error, reason)}
22       });
23     end
24
25     it 'should result in a deleted document'
26       db.removeDoc({_id : "345", _rev : saved_doc.rev}, {
27         success: function(resp){
28           db.openDoc("345", {
29             error: function(status, error, reason){
30               status.should.eql 404
31               error.should.eql "not_found"
```

```
32              reason.should.eql "deleted"
33            },
34            success: function(resp){successCallback(resp)}
35          });
36        },
37        error: function(status, error, reason){errorCallback(status, error, reason)}
38      });
39    end
40
41    it 'should return ok true, the ID and the revision of the deleted document'
42      db.removeDoc({_id : "345", _rev : saved_doc.rev}, {
43        success: function(resp){
44          resp.ok.should.be_true
45          resp.id.should.eql "345"
46          resp.rev.should.be_a String
47          resp.rev.length.should.be_at_least 30
48        },
49        error: function(status, error, reason){errorCallback(status, error, reason)}
50      });
51    end
52
53    it 'should record the revision in the deleted document'
54      db.removeDoc({_id : "345", _rev : saved_doc.rev}, {
55        success: function(resp){
56          db.openDoc("345", {
57            rev: resp.rev,
58            success: function(resp2){
59              resp2._rev.should.eql resp.rev
60              resp2._id.should.eql resp.id
61              resp2._deleted.should.be_true
62            },
63            error: function(status, err, rsn){errorCallback(status, err, rsn)}
64          });
65        },
66        error: function(status, error, reason){errorCallback(status, error, reason)}
67      });
68    end
69
70    it 'should alert with an error message prefix'
71      db.removeDoc({_id: "asdf"});
72      alert_msg.should.match /The document could not be deleted/
73    end
74  end
75
76  \\ [...]
77 end
```

Listing A.9: Auszug aus `/share/www/spec/jquery_couch_3_spec.js`

A.5.9 Screenshots der AWS Management Console

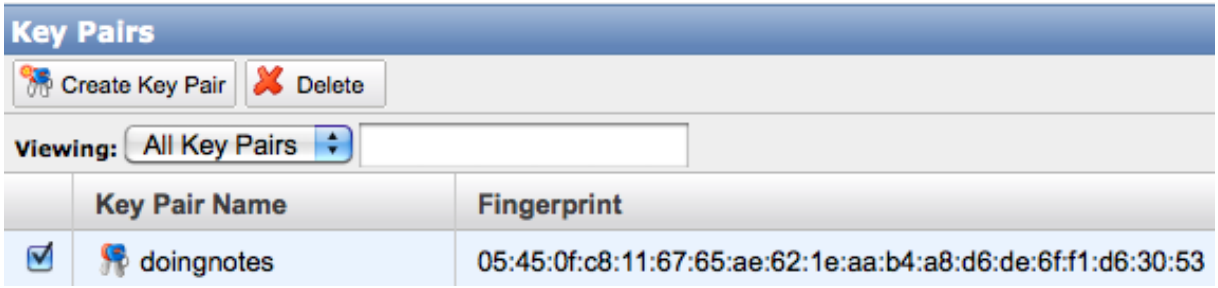

Abbildung A.10: AWS: Schlüsselpaar für die Authentifizierung

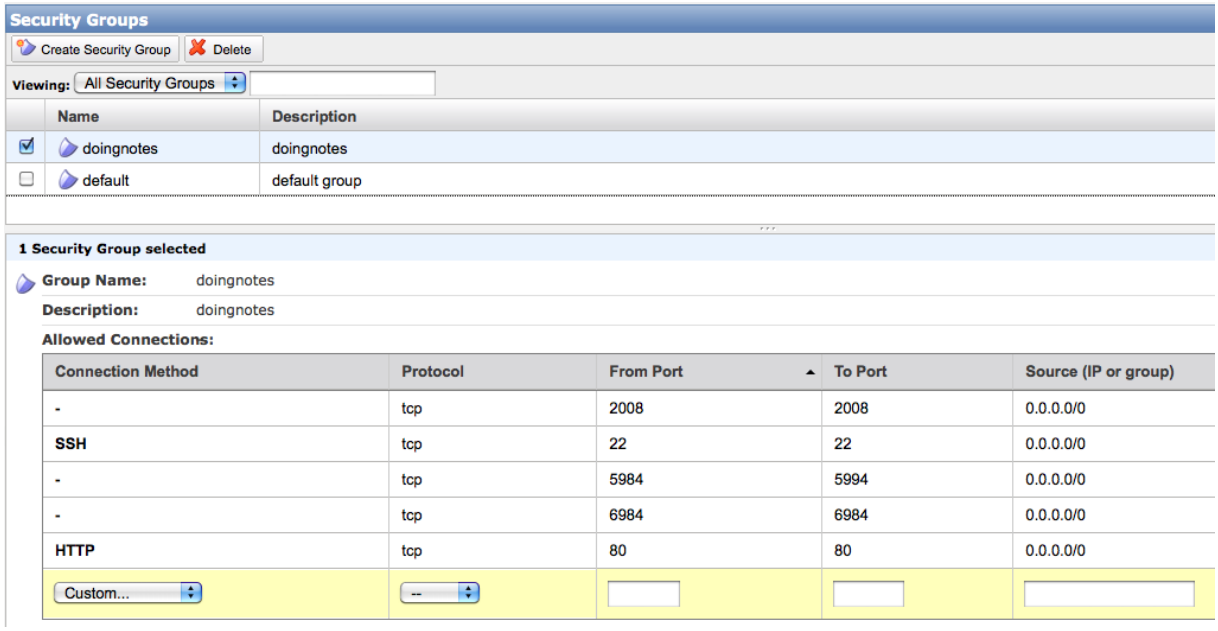

Abbildung A.11: AWS: Freigeben der Ports über die Security Group

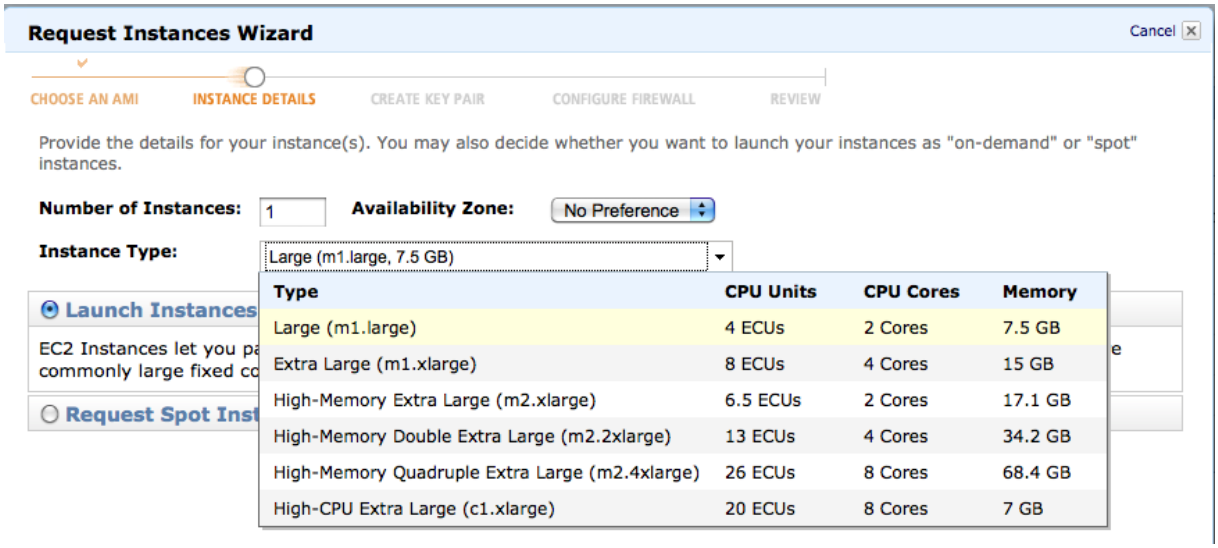

Abbildung A.12: AWS: Auswahl der Kapazitäten der Instanz

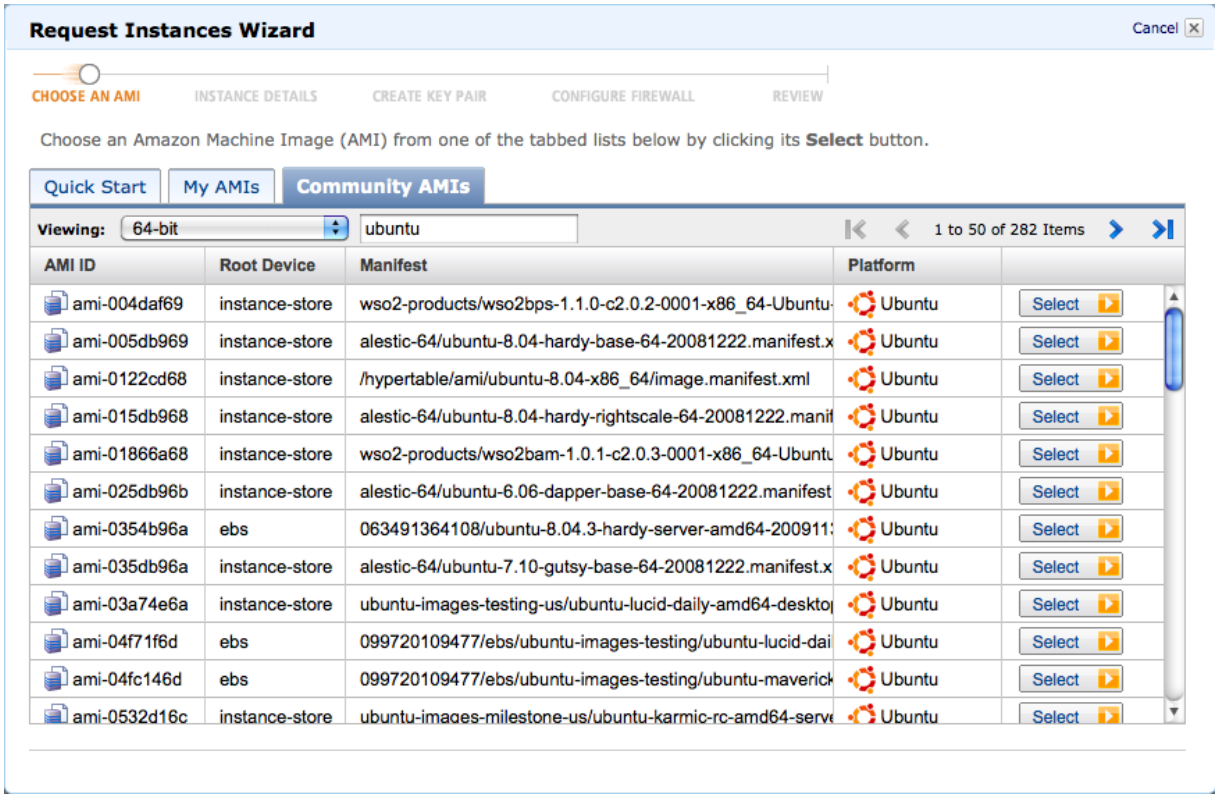

Abbildung A.13: AWS: Auswahl des Amazon Machine Images

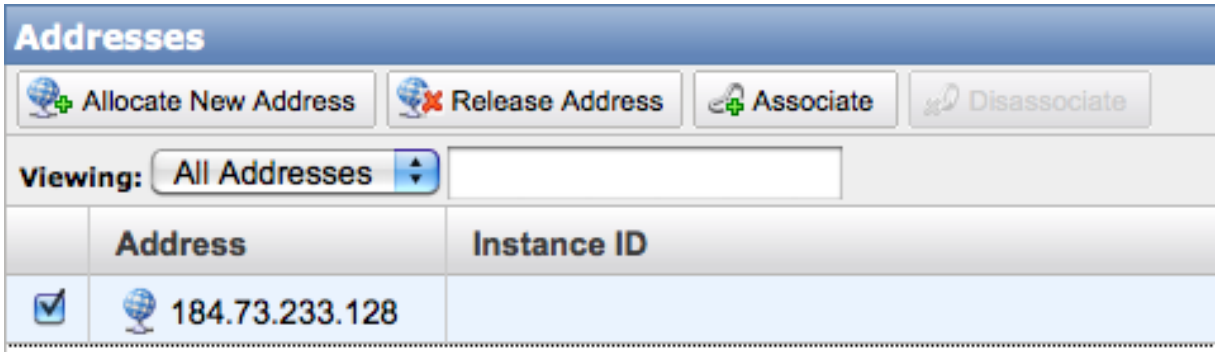

Abbildung A.14: AWS: Einrichten einer Elastic IP

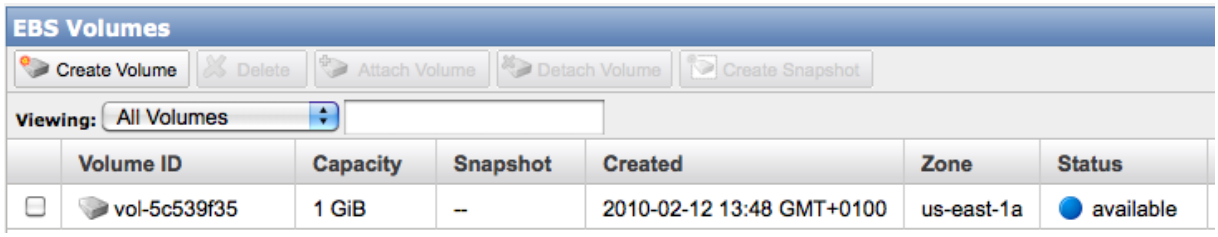

Abbildung A.15: AWS: Einrichten eines EBS Volumes

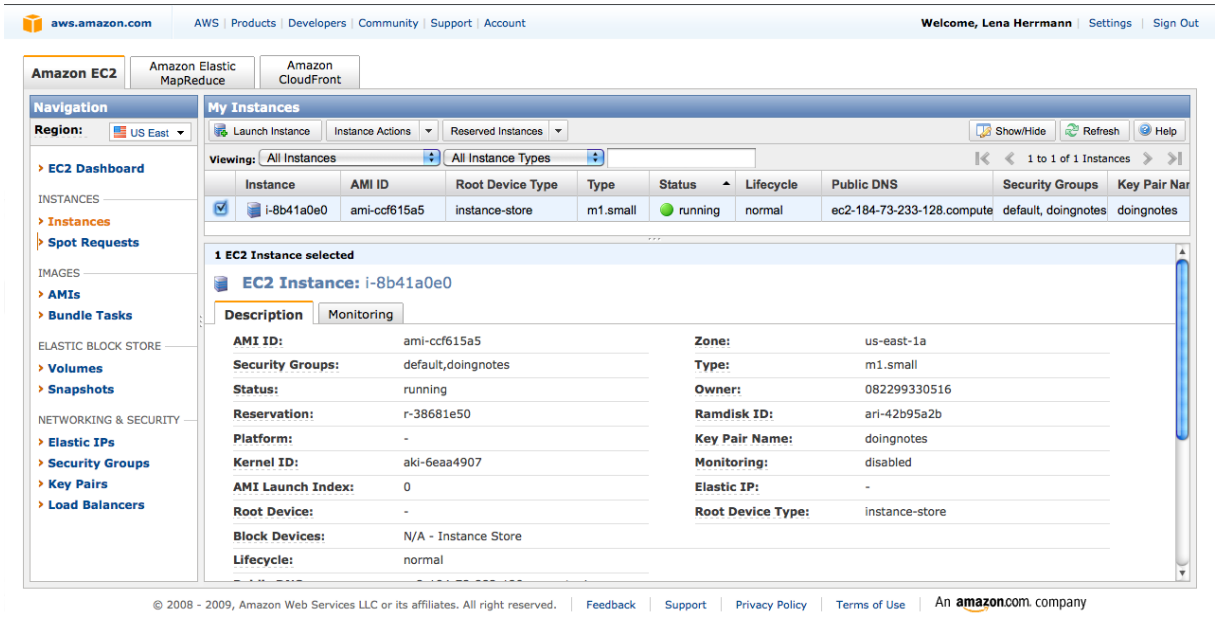

Abbildung A.16: AWS: Die laufende EC2-Instanz

A.6 Inhalt der CD-ROM

Nachstehend ist der Inhalt der der Diplomarbeit beiliegenden CD-ROM aufgeführt.

Die Datei *Diplom_Lena_Herrmann.pdf* enthält die schriftliche Ausarbeitung der Diplomarbeit.

Im Verzeichnis *Implementierung* befindet sich der Quelltext der implementierten Anwendung.

Das Verzeichnis *Software* enthält die für den Betrieb der Anwendung nötige Software.

Verzeichnisse

Literaturverzeichnis

[Ada07] Adams, Tom: *Better Testing Through Behaviour*. Talk at Open Source Developers' Conference in Brisbane, Australia, November 2007.

[And10] Anderson, J. Chris; Lehnardt, Jan; Slater, Noah: *CouchDB: The Definitive Guide*. O'Reilly Media, Februar 2010.

[Ass98] Assfalg, Rolf; Goebels, Udo; Welter, Heinrich: *Internet Datenbanken*. Addison-Wesley, 1998.

[Bau10] Baun, Christian; Kunze, Marcel; Nimis, Jens; Tai, Stefan: *Cloud Computing. Web-basierte dynamische IT-Services*. Springer-Verlag, 2010.

[Bay08] Bayer, Rudolf: *B-tree and UB-tree*. Scholarpedia, 3(11):7742, 2008.

[Ben04] Bengel, Günther: *Verteilte Systeme. 3. Auflage*. Vieweg, 2004.

[Ber81] Bernstein, Philip A.; Goodman, Nathan: *Concurrency Control in Distributed Database Systems*. ACM Comput. Surv., 13(2):185–221, 1981.

[Ble08] Bleek, Wolf-Gideon; Wolf, Henning: *Agile Softwareentwicklung*. dpunkt.verlag, Februar 2008.

[Bre00] Brewer, Eric A.: *Towards robust distributed systems (abstract)*. *PODC*, 7, 2000.

[Cha06] Chang, Fay et.al.: *Bigtable: A Distributed Storage System for Structured Data*. OSDI'06: Seventh Symposium on Operating System Design and Implementation, 2006.

[Che07] Chess, Brian; Tsipenyuk O'Neil, Yekaterina; West, Jacob: *Javascript Hijacking*. März 2007.

[Cod83] Codd, Edgar Frank: *A relational model of data for large shared data banks*. Communications of the ACM, 26(1):64–69, 1983.

[Cor01] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford: *Introduction to Algorithms*. The MIT Press, 2nd Revised edition , September 2001.

[Cou05] Coulouris, George; Dollimore, Jean; Kindberg, Tim: *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science Series)*. Addison Wesley, May 2005.

[Cro08] Crockford, Douglas: *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

[Edl95] Edlich, Stefan: *Kooperationsmechanismen synchroner Groupwaresysteme. Entwurf, Realisierung und Einsatz eines Werkzeugsatzes für replizierte Softwarekooperation*. Technische Universität Berlin, 1995.

[Ell89] Ellis, Clarence A.; Gibbs, Simon John D.: *Concurrency control in groupware systems*. SIGMOD Rec., 18(2):399–407, 1989.

[Fie96] Fielding, R.; Frystyk, H.; Berners-Lee, T.; Gettys, J.; Mogul, Jeffrey C.: *Hypertext Transfer Protocol - HTTP/1.1*, 1996.

[Fie00] Fielding, Roy: *Architectural Styles and the Design of Network-based Software Architectures*. , University of California, 2000.

[Fis08] Fischer, Jens-Christian: *Professionelle Webentwicklung mit Ruby on Rails 2*. mitp, 2008.

[Gil02] Gilbert, Seth; Lynch, Nancy: *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. SIGACT News, 33(2):51–59, 2002.

[GJ05] Godwin-Jones, Robert: *Ajax and Firefox: New Web Applications and Browsers*. Language Learning and Technology, 9(2):8–12, 2005.

[Gra96] Gray, Jim; Helland, Pat; O'Neil, Patrick; Shasha, Dennis: *The dangers of replication and a solution*. *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, 173–182, New York, NY, USA, 1996. ACM.

[Guy98] Guy, Richard; Reiher, Peter; Ratner, David; Gunter, Michial; Ma, Wilkie; Popek, Gerald: *Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication*. 1998.

[Gü04] Güting, Ralf Hartmut; Dieker, Stefan: *Datenstrukturen und Algorithmen*. Teubner Verlag, Dezember 2004.

[Ham08] Hamill, Paul: *Unit Test Frameworks*. O'Reilly Media, November 2008.

[Hes92] Hesse, Wolfgang; Merbeth, Günther; Frölich, Rainer: *Software-Entwicklung. Vorgehensmodelle, Projektführung, Produktverwaltung*. R. Oldenbourg Verlag, 1992.

[Hup04] Hupfeld, Felix: *Log-Structured Storage for Efficient Weakly-Connected Replication*. ICDCS Workshops, 458–463, März 2004.

[Hup09] Hupfeld, Felix: *Causal Weak-Consistency Replication – A Systems Approach*. , Humboldt-Universität zu Berlin, Januar 2009.

[Hä01] Härder, Theo; Rahm, Erhard: *Datenbanksysteme: Konzepte und Techniken der Implementierung, 2. Auflage*. Springer, 2001.

[Ini10] Initiative D21: *(N)ONLINER Atlas 2010. Eine Topographie des digitalen Grabens durch Deutschland. Nutzung und Nichtnutzung des Internets, Strukturen und regionale Verteiltung.* Juli 2010.

[Kar95] Karsten, Helena: *"It's like everyone working around the same desk": organisational readings of Lotus Notes.* Scand. J. Inf. Syst., 7(1):3–32, 1995.

[Kaw88] Kawell,Jr., Leonard; Beckhardt, Steven; Halvorsen, Timothy; Ozzie, Raymond; Greif, Irene: *Replicated document management in a group communication system. CSCW '88: Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, 395. ACM, 1988.

[Kli96] Klieb, Leslie: *Distributed disconnected databases. SAC '96: Proceedings of the 1996 ACM symposium on Applied Computing*, 322–326. ACM, 1996.

[Knu97] Knuth, Donald: *The art of computer programming vol 1. Fundamental Algorithms, Dritte Auflage.* Addison-Wesley, 1997.

[Kot99] Kotz, D, ; Gray R.S.: *Mobile Agents and the Future of the Internet.* 7–13, August 1999.

[Kra09a] Krasnova, Hanna; Günther, Oliver; Spiekermann, Sarah; Koroleva, Ksenia: *Privacy Concerns and Identity in Social Networks (submitted).* Juli 2009.

[Kra09b] Krasnova, Hanna; Spiekermann, Sarah; Thomas, Hildebrand: *Online Social Networks: Why so we disclose (submitted).* Juli 2009.

[KT04] Khare, Rohit Richard N. Taylor: *Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems. ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, 428–437, Washington, DC, USA, 2004. IEEE Computer Society.

[Lam78] Lamport, Leslie: *Time, Clocks, and the Ordering of Events in a Distributed System.* Communications of the ACM, 21(7), Juli 1978.

[Lam01] Lamport, Leslie: *Paxos Made Simple.* ACM SIGACT News, 32(4):18–25, December 2001.

[Leh08] Lehnardt, Jan: *Introduction into CouchDB.* Talk at BBC London, August 2008.

[Leh09] Lehnardt, Jan: *Peer-to-peer Applications with CouchDB.* Talk at NoSQL Berlin, Oktober 2009.

[Leh10] Lehnardt, Jan: *Making Software for Humans - CouchDB and the Usable Peer-To-Peer Web.* Talk at Berlin Buzzwords, June 2010.

[Mat07] Matthiessen, Günter; Unterstein, Michael: *Relationale Datenbanken und Standard-SQL - Konzepte der Entwicklung und Anwendung*. Addison-Wesley, Dezember 2007.

[Meh97] Mehta, Manish; DeWitt, David J.: *Data placement in shared-nothing parallel database systems*. The VLDB Journal, 6(1):53–72, 1997.

[Mil68] Miller, Robert B.: *Response time in man-computer conversational transactions*. Proceedings of the AFIPS Fall Joint Computer Conference, 33:267–277, 1968.

[Mon01] Montoya-Weiss, Mitzi M.; Massey, Anne P.; Song, Michael: *Getting It Together: Temporal Coordination and Conflict Management in Global Virtual Teams*. The Academy of Management Journal, 44(6):1251–1262, Dezember 2001.

[Mos09] Moser, Monika: *Consistency in Distributed Key/Value Stores*. Talk at NoSQL Berlin, Oktober 2009.

[Nie93] Nielsen, Jakob: *Usability Engineering (Interactive Technologies)*. Morgan Kaufmann, November 1993.

[Orl92] Orlikowski, Wanda: *Learning from NOTES: Organizational Issues in Groupware Implementation*. 134, MIT Center for Coordination Science, August 1992.

[Par10] Parveen, Suraiya; Tanweer, Safrad: *A Study on Effect of Replication on Databases*. *Proceedings of the 4th National Conference; INDIACom-2010*, 483–490, Februar 2010.

[Pau05] Paulson, Linda Dailey: *Building Rich Web Applications with Ajax*. Computer, 38(10):14–17, 2005.

[Pea80] Pease, Marshall; Shostak, Robert; Lamport, Leslie: *Reaching Agreement in the Presence of Faults*. J. ACM, 27(2):228–234, 1980.

[Qia09] Qian, Ling; Luo, Zhiguo; Du, Yujian; Guo, Leitao: *Cloud Computing: An Overview*. *Cloud Computing - First International Conference, CloudCom 2009*, 626–631. Springer, Dezember 2009.

[Ray07] Raymond, Scott: *Ajax on Rails*. O'Reilly Media, Inc., 2007.

[Sai05] Saito, Yasushi; Shapiro, Marc: *Optimistic replication*. ACM Comput. Surv., 37(1):42–81, 2005.

[Ski07] Skiadas, C.; Kjosmoen, T.: *LATEXing with TextMate*. The PracTEX Journal, (3), 2007.

[Spr97] Spreitzer, Mike; Theimer, Marvin M.; Petersen, Karin; J, Alan; Terry, Douglas B.: *Dealing with Server Corruption in Weakly Consistent, Replicated Data Systems*. *In Proc. of ACM/IEEE MobiCom Conf*, 234–240, 1997.

[Sto86] Stonebreaker, Michael: *The case for shared nothing*. IEEE Database Engineering Bulletin, 9(1):4–9, 1986.

[Tan07] Tanenbaum, Andrew S.; van Steen, Maarten: *Verteilte Systeme*. Pearson Studium, 2., Aufl. , 2007.

[Van96] Vandenbosch, Betty; Ginzberg, Michael J.: *Lotus notes®and collaboration: plus ça change...* J. Manage. Inf. Syst., 13(3):65–81, 1996.

[Var09] Varley, Ian Thomas: *No Relation: The Mixed Blessings of Non-Relational Databases*. , University of Texas at Austin, August 2009.

[Vog09] Vogels, Werner: *Eventually Consistent. Building reliable distributed systems at a world-wide scale demands trade-offs between consistency and availability*. Communications of the ACM, 52(1), Januar 2009.

[Yan07] Yang, Hung-chih; Dasdan, Ali; Hsiao, Ruey-Lung; Parker, D. Stott: *Map-reduce-merge: simplified relational data processing on large clusters. SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 1029–1040, New York, NY, USA, 2007. ACM.

Internetquellen

[10g10] 10GEN: *mongoDB Replication*. http://www.mongodb.org/display/DOCS/Replication, 2010. zuletzt abgerufen am 16.06.2010.

[Alf08] ALFRESCO SOFTWARE, INC: *The Open Source Barometer III*. hhttp://www.alfresco.com/community/barometer/, Februar 2008. zuletzt abgerufen am 26.06.2010.

[All10] ALLSOPP, JOHN: *The State of Web Development 2010*. http://www.webdirections.org/sotw10/, April 2010. zuletzt abgerufen am 14.06.2010.

[Amaa] AMAZON WEB SERVICES LLC: *Amazon EC2-Instanztypen*. http://aws.amazon.com/de/ec2/instance-types/. zuletzt abgerufen am 20.06.2010.

[Amab] AMAZON WEB SERVICES LLC: *AWS Simple Monthly Calculator*. http://calculator.s3.amazonaws.com/calc5.html. zuletzt abgerufen am 20.06.2010.

[Amb] AMBLER, SCOTT W.: *Disciplined Agile Software Development: Definition*. http://www.agilemodeling.com/essays/agileSoftwareDevelopment.htm. zuletzt abgerufen am 20.06.2010.

[And10] ANDERSON, J. CHRIS: *Evently Docs*. http://github.com/jchris/toast/blob/master/vendor/couchapp/docs/evently.md, 2010. zuletzt abgerufen am 07.07.2010.

[Apa09a] APACHE SOFTWARE FOUNDATION: *The Apache Cassandra Project*. http://cassandra.apache.org/, 2009. zuletzt abgerufen am 06.07.2010.

[Apa09b] APACHE SOFTWARE FOUNDATION: *Apache CouchDB: Technical Overview*. http://couchdb.apache.org/docs/overview.html, 2009. zuletzt abgerufen am 26.06.2010.

[Apa09c] APACHE SOFTWARE FOUNDATION: *Apache CouchDB: The CouchDB Project*. http://couchdb.apache.org/, 2009. zuletzt abgerufen am 05.06.2010.

[Apa09d] APACHE SOFTWARE FOUNDATION: *Graduated from incubation*. http://incubator.apache.org/projects/, 2009. zuletzt abgerufen am 24.06.2010.

[Apa10a] APACHE GITHUB REPOSITORY: *Apache CouchDB Commit: Show conflicts in changes filters. Patch by Lena Herrmann*. http://github.com/apache/couchdb/commit/5a87b852a72cc36d4d7b64271ed542ce2a1befe0, 2010. zuletzt abgerufen am 04.06.2010.

[Apa10b] Apache Software Foundation: *Apache Subversion*. http://subversion.apache.org/, 2010. zuletzt abgerufen am 16.06.2010.

[Apa10c] Apache Software Foundation: *CouchDB Github Repository*. http://github.com/apache/couchdb, 2010. zuletzt abgerufen am 06.06.2010.

[Apa10d] Apache Software Foundation: *CouchDB Subversion Repository*. https://svn.apache.org/repos/asf/couchdb/, 2010. zuletzt abgerufen am 06.06.2010.

[Apa10e] Apache Software Foundation: *JIRA-Ticket: Add Bulk Deletion to jquery.couch.js*. https://issues.apache.org/jira/browse/COUCHDB-612, 2010. zuletzt abgerufen am 04.06.2010.

[Apa10f] Apache Software Foundation: *JIRA-Ticket: Add tests for Javascript HTTP API*. https://issues.apache.org/jira/browse/COUCHDB-783, 2010. zuletzt abgerufen am 04.06.2010.

[Apa10g] Apache Software Foundation: *JIRA-Ticket: Make bulkSave actually save docs*. https://issues.apache.org/jira/browse/COUCHDB-634, 2010. zuletzt abgerufen am 04.06.2010.

[Apa10h] Apache Software Foundation: *JIRA-Ticket: Make the changes_filter use _conflicts field*. https://issues.apache.org/jira/browse/COUCHDB-630, 2010. zuletzt abgerufen am 04.06.2010.

[Apa10i] Apache Software Foundation: *Welcome to HBase*. http://hbase.apache.org/, 2010. zuletzt abgerufen am 06.07.2010.

[Ape09] Apestaart, Thomas: *Building erlang for the N900*. http://thomas.apestaart.org/log/?p=1086, Dezember 2009. zuletzt abgerufen am 10.06.2010.

[Ape10] Apestaart, Thomas: *desktopcouch on N900*. http://thomas.apestaart.org/log/?p=1106, Januar 2010. zuletzt abgerufen am 10.06.2010.

[App10] Apple Inc.: *Common QA for Bonjour*. http://developer.apple.com/mac/library/qa/qa2010/qa1690.html, Mai 2010. zuletzt abgerufen am 09.07.2010.

[Bad08] Bader, Chrys: *jQuery Plugins: Auto Growing Textareas*. http://plugins.jquery.com/project/autogrow, januar 2008. zuletzt abgerufen am 05.07.2010.

[Bak] Bakken, Jarib: *Celerity*. http://celerity.rubyforge.org/. zuletzt abgerufen am 21.06.2010.

[Bar09] Barreto, Charlton: *NoSQL: leading Cloud's "NoBah" movement?* http://charltonb.typepad.com/weblog/2009/07/

nosql-leading-clouds-nobah-movement.html, Juli 2009. zuletzt abgerufen am 09.05.2010.

[Bas10] Basho: *Riak: An Open Source Internet-Scale Data Store*. http://wiki.basho.com/display/RIAK/Riak, 2010. zuletzt abgerufen am 16.06.2010.

[Bec01] Beck, Kent; Beedle, Mike; van Bennekum Arie; Cockburn Alistair et. al.: *Manifesto for Agile Software Development*. http://www.agilemodeling.com/essays/agileSoftwareDevelopment.htm, Februar 2001. zuletzt abgerufen am 20.06.2010.

[Bec07] Beck, Kent: *Three Rivers Institute: Test-Driven Development Violates the Dichotomies of Testing*. http://www.threeriversinstitute.org/Testing%20Dichotomies%20and%20TDD.htm, Juni 2007. zuletzt abgerufen am 22.06.2010.

[Bel10] Belomestnykh, Olga: *A rebuilt, more real time Google documents*. http://googledocs.blogspot.com/2010/04/rebuilt-more-real-time-google-documents.html, April 2010. zuletzt abgerufen am 06.06.2010.

[Ben06] Bengtsson, Anders: *Madeleine*. http://madeleine.rubyforge.org/, 2006. zuletzt abgerufen am 12.06.2010.

[Cha10] Chacon, Scott: *git - the fast version control system*. http://git-scm.com/, 2010. zuletzt abgerufen am 16.06.2010.

[Che] Chelimsky, David: *RSpec*. http://rspec.info/. zuletzt abgerufen am 21.06.2010.

[Che10] Chesneau, Benoit: *Couchapp Github Repository*. http://github.com/couchapp/couchapp, 2010. zuletzt abgerufen am 05.06.2010.

[Cora] Corporation, Oracle: *MySQL*. http://www.mysql.com/?bydis_dis_index=1. zuletzt abgerufen am 22.06.2010.

[Corb] Corporation, Oracle: *MySQL 5.5 Replication and AUTO_INCREMENT*. http://dev.mysql.com/doc/refman/5.5/en/replication-features-auto-increment.html. zuletzt abgerufen am 26.06.2010.

[Corc] Corporation, Oracle: *MySQL 5.5 Replication Implementation*. http://dev.mysql.com/doc/refman/5.5/en/replication-implementation.html. zuletzt abgerufen am 26.06.2010.

[Cord] Corporation, Oracle: *MySQL Cluster Replication*. http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-replication.html. zuletzt abgerufen am 22.06.2010.

[Cou09] CouchDB Wiki: *Designing an application to work with replication*. http://wiki.apache.org/couchdb/How_to_design_for_replication, 2009. zuletzt abgerufen am 12.06.2010.

[cou10a] couchdb-lounge wiki: *Setting up two couch instances*. http://code.google.com/p/couchdb-lounge/wiki/SettingUpTwoCouchInstances, 2010. zuletzt abgerufen am 05.06.2010.

[Cou10b] CouchDB Wiki: *CouchDB Windows_binary_installer*. http://wiki.apache.org/couchdb/Windows_binary_installer, 2010. zuletzt abgerufen am 06.06.2010.

[Cou10c] CouchDB Wiki: *Installation of CouchDB and Dependencies*. http://wiki.apache.org/couchdb/Installation, 2010. zuletzt abgerufen am 05.06.2010.

[Cro06] Crockford, Douglas: *Network Working Group, Request for Comments 4627*. http://www.ietf.org/rfc/rfc4627.txt, Juli 2006. zuletzt abgerufen am 05.06.2010.

[Cro10] Crockford, Douglas: *JSON*. http://www.json.org/, 2010. zuletzt abgerufen am 05.06.2010.

[Dro10] Dropbox: *Dropbox*. https://www.dropbox.com/, 2010. zuletzt abgerufen am 16.06.2010.

[Eri10] Ericsson: *Open Source Erlang*. http://www.erlang.org/, 2010. zuletzt abgerufen am 06.06.2010.

[Eym08] Eymann, Torsten: *Cloud Computing*. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/uebergreifendes/Kontext-und-Grundlagen/Markt/Softwaremarkt/Geschaftsmodell-%28fur-Software-und-Services%29/Cloud-Computing/index.html?searchterm=Cloud+, September 2008. zuletzt abgerufen am 19.06.2010.

[Fel09] Fellowship of the packaging: *Easy Install - Distribute documentation*. http://packages.python.org/distribute/easy_install.html, 2009. zuletzt abgerufen am 06.06.2010.

[Fou10] Foundation, Etherpad: *Etherpad - Live Collaborative Text*. http://www.etherpad.org/, 2010. zuletzt abgerufen am 07.06.2010.

[Gar] Gargoyle Software: *HtmlUnit*. http://htmlunit.sourceforge.net/. zuletzt abgerufen am 21.06.2010.

[Gar05] Garrett, Jesse James: *Ajax: A New Approach to Web Applications*. http://www.adaptivepath.com/ideas/essays/archives/000385.php, Februar 2005. zuletzt abgerufen am 05.06.2010.

[Gar09] Gartner: *Gartner's 2009 Hype Cycle Special Report Evaluates Maturity of 1,650 Technologies*. http://www.gartner.com/it/page.jsp?id=1124212, August 2009. zuletzt abgerufen am 17.06.2010.

[Goo10] Google: *Google Wave*. http://wave.google.com/about.html, 2010. zuletzt abgerufen am 07.06.2010.

[Groa] Group, PostgreSQL GLobal Development: *PostgreSQL*. http://www.postgresql.org/. zuletzt abgerufen am 22.06.2010.

[Grob] Group, PostgreSQL GLobal Development: *Replication, Clustering, and Connection Pooling*. http://wiki.postgresql.org/index.php?title=Replication%2C_Clustering%2C_and_Connection_Pooling&oldid=10880. zuletzt abgerufen am 22.06.2010.

[Hel] Hellesøy, Aslak: *Cucumber. Behaviour Driven Development with elegance and joy*. http://cukes.info/. zuletzt abgerufen am 21.06.2010.

[Hel08] Helmkamp, Bryan: *Webrat - Ruby Acceptance Testing for Web applications*. http://github.com/brynary/webrat, 2008. zuletzt abgerufen am 06.07.2010.

[Her10] Herrmann, Lena: *JSpec - JavaScript Unit testing how it should be*. http://lenaherrmann.net/2010/01/04/jspec-javascript-unit-testing-how-it-should-be, Januar 2010. zuletzt abgerufen am 21.05.2010.

[Hic10a] Hickson, Ian: *HTML5 - A vocabulary and associated APIs for HTML and XHTML*. http://dev.w3.org/html5/spec/Overview.html, 2010. zuletzt abgerufen am 06.06.2010.

[Hic10b] Hickson, Ian: *HTML5 - A vocabulary and associated APIs for HTML and XHTML - Embedding custom non-visible data*. http://dev.w3.org/html5/spec/Overview.html#embedding-custom-non-visible-data, 2010. zuletzt abgerufen am 06.06.2010.

[Ho,08] Ho, Ricky: *CouchDB Implementation*. http://horicky.blogspot.com/2008/10/couchdb-implementation.html, Oktober 2008. zuletzt abgerufen am 18.06.2010.

[Hol10] Holowaychuk, TJ: *JSpec - JavaScript Testing Framework*. http://visionmedia.github.com/jspec/, 2010. zuletzt abgerufen am 06.06.2010.

[Hyp09] Hypertable: *Hypertable - An Open Source, High Performance, Scalable Database*. http://hypertable.org/, 2009. zuletzt abgerufen am 06.07.2010.

[IDG10] IDG BUSINESS MEDIA GMBH: *Firebug - ein Muss für Web-Entwickler*. http://www.computerwoche.de/software/software-infrastruktur/1934278/, April 2010. zuletzt abgerufen am 09.06.2010.

[Joh09] JOHNSON, NICK: *Damn Cool Algorithms: Log structured storage*. http://blog.notdot.net/2009/12/Damn-Cool-Algorithms-Log-structured-storage, Dezember 2009. zuletzt abgerufen am 17.05.2010.

[jQu10] JQUERY PROJECT: *jQuery: The Write Less, Do More, JavaScript Library*. http://www.jquery.org/, 2010. zuletzt abgerufen am 05.06.2010.

[Kap07] KAPLAN-MOSS, JACOB: *Of the Web*. http://jacobian.org/writing/of-the-web/, Oktober 2007. zuletzt abgerufen am 07.07.2010.

[Kat10] KATZ, DAMIEN: *Interview with Damien Katz – Apache CouchDB*. http://howsoftwareisbuilt.com/2010/06/18/interview-with-damien-katz-apache-couchdb/, Juni 2010. zuletzt abgerufen am 24.06.2010.

[Kla10] KLAMPAECKEL, TILL: *A toolchain for CouchDB Lounge*. http://till.klampaeckel.de/blog/archives/84-A-toolchain-for-CouchDB-Lounge.html, Januar 2010. zuletzt abgerufen am 18.06.2010.

[Lan] LANG, ALEXANDER: *Culerity*. http://github.com/langalex/culerity. zuletzt abgerufen am 21.06.2010.

[Lan09a] LANDAU, BRIAN: *Benchmarking Javascript Templating Libraries*. http://www.viget.com/extend/benchmarking-javascript-templating-libraries/, Dezember 2009. zuletzt abgerufen am 06.06.2010.

[Lan09b] LANG, ALEXANDER: *Culerity = Full Stack Rails Testing with Cucumber and Celerity*. http://upstre.am/2009/01/28/culerity-full-stack-rails-testing-with-cucumber-and-celerity/, Januar 2009. zuletzt abgerufen am 21.06.2010.

[Law10] LAW, PAUL: *Tools: Outlining via OmniOutliner*. http://lawpaul.wordpress.com/2010/02/20/tools-outlining-via-omnioutliner/, Februar 2010. zuletzt abgerufen am 10.06.2010.

[Lea05] LEACH, PAUL J.: *RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace*. http://tools.ietf.org/html/rfc4122, Juli 2005. zuletzt abgerufen am 07.07.2010.

[Lee09a] LEEDS, RANDALL: *couchdb-lounge wiki*. http://wiki.github.com/tilgovi/couchdb-lounge/, 2009. zuletzt abgerufen am 18.06.2010.

[Lee09b] Leeds, Randall: *Partitioning and Clustering Support for CouchDB. Proposal for Google Summer of Code 2009*. http://socghop.appspot.com/document/show/user/rleeds/couchdb_cluster, April 2009. zuletzt abgerufen am 18.06.2010.

[Lee10] Leeds, Randall: *Lounge - a proxy-based partitioning/clustering framework for CouchDB*. http://tilgovi.github.com/couchdb-lounge/, 2010. zuletzt abgerufen am 16.06.2010.

[Lef02] Lefkowitz, Glyph; Zadka, Moshe: *The Twisted Network Framework*. http://www.python.org/workshops/2002-02/papers/09/index.htm, Februar 2002. zuletzt abgerufen am 18.06.2010.

[Leh10a] Lehnardt, Jan: *Couchio: What's new in Apache CouchDB 0.11 — Part Three: New Features in Replication*. http://blog.couch.io/post/468392274/whats-new-in-apache-couchdb-0-11-part-three-new, März 2010. zuletzt abgerufen am 07.07.2010.

[Leh10b] Lehnardt, Jan: *Couchio: Mustache.js*. http://blog.couch.io/post/622014913/mustache-js, Mai 2010. zuletzt abgerufen am 06.06.2010.

[Leh10c] Lehnardt, Jan: *mustache.js Github Repository*. http://github.com/janl/mustache.js, 2010. zuletzt abgerufen am 06.06.2010.

[Leh10d] Lehnardt, Jan; Grosenbach, Geoffrey; Gretar, Jon: *CouchDBX Github Repository*. http://janl.github.com/couchdbx/, 2010. zuletzt abgerufen am 05.06.2010.

[Len07] Lenz, Christopher: *CouchDB „Joins"*. http://www.cmlenz.net/archives/2007/10/couchdb-joins, Oktober 2007. zuletzt abgerufen am 13.06.2010.

[Len09] Lennon, Joe: *Exploring CouchDB. A document-oriented database for Web applications*. http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html, März 2009. zuletzt abgerufen am 24.06.2010.

[Mac05] Macworld: *The Omni Group OmniOutliner 3 Professional Utility Software Review*. http://www.macworld.com/reviews/product/405814/review/omnioutliner_3_professional.html, Juni 2005. zuletzt abgerufen am 15.06.2010.

[Mac10] MacroMates: *textmate - the missing editor*. http://macromates.com/, 2010. zuletzt abgerufen am 06.06.2010.

[Man09] Manjoo, Farhad: *It's Just Fancy Talk - The Google Wave chatting tool is too complicated for its own good*. http://www.slate.com/id/2232311/pagenum/all/, Oktober 2009. zuletzt abgerufen am 07.05.2010.

[Mat10] Matsumoto, Yukihiro: *Ruby - A Programmer's best friend*. http://www.ruby-lang.org/de/, 2010. zuletzt abgerufen am 06.06.2010.

[Max06] Maxia, Giuseppe: *Advanced MySQL Replication Techniques*. http://onlamp.com/pub/a/onlamp/2006/04/20/advanced-mysql-replication.html, April 2006. zuletzt abgerufen am 22.06.2010.

[Mem09] MemcacheDB: *MemcacheDB: A distributed key-value storage system designed for persistent*. http://memcachedb.org/, 2009. zuletzt abgerufen am 06.07.2010.

[Mil07] Miller, Jeremy D.: *BDD, TDD, and the other Double D's*. http://codebetter.com/blogs/jeremy.miller/archive/2007/09/06/bdd-tdd-and-the-other-double-d-s.aspx, September 2007. zuletzt abgerufen am 21.06.2010.

[Mil10] Miller, Aaron: *Android CouchDB releases*. http://apage43.github.com/, 2010. zuletzt abgerufen am 10.06.2010.

[Mon] Montoya, Christian: *Blueprint: A CSS Framework*. http://www.blueprintcss.org/. zuletzt abgerufen am 04.07.2010.

[Moza] Mozilla Foundation: *Add-ons für Firefox - Beliebte Add-ons*. https://addons.mozilla.org/de/firefox/browse/type:1/cat:all?sort=popular. zuletzt abgerufen am 09.06.2010.

[Mozb] Mozilla Foundation: *Firebug Statistics*. https://addons.mozilla.org/en-US/firefox/statistics/addon/1843. zuletzt abgerufen am 09.06.2010.

[Mozc] Mozilla Foundation: *Rhino: JavaScript for Java*. http://www.mozilla.org/rhino/. zuletzt abgerufen am 22.06.2010.

[Moz10a] Mozilla Foundation: *Firebug - Web Development Evolved*. http://getfirebug.com/, 2010. zuletzt abgerufen am 06.06.2010.

[Moz10b] Mozilla Foundation: *Firefox 3.6 – Versionshinweise*. http://www.mozilla-europe.org/de/firefox/3.6/releasenotes/, Januar 2010. zuletzt abgerufen am 07.05.2010.

[Moz10c] Mozilla Foundation: *SpiderMonkey (JavaScript-C) Engine*. http://www.mozilla.org/js/spidermonkey/, 2010. zuletzt abgerufen am 06.06.2010.

[Moz10d] Mozilla Foundation: *Webbrowser Firefox*. http://getfirefox.com/, 2010. zuletzt abgerufen am 05.06.2010.

[mus10] *Mustache*. http://mustache.github.com/, 2010. zuletzt abgerufen am 06.06.2010.

[NOR] NORD Software Consulting: *Classification of HTTP-based APIs*. http://nordsc.com/ext/classification_of_http_based_apis.html. zuletzt abgerufen am 06.06.2010.

[NOR10] NORD SOFTWARE CONSULTING: *Classifying the CouchDB API*. http://www.slate.com/id/2232311/pagenum/all/, Februar 2010. zuletzt abgerufen am 06.06.2010.

[Omn10] OMNI GROUP: *OmniOutliner 3*. http://www.omnigroup.com/products/omnioutliner/, 2010. zuletzt abgerufen am 10.06.2010.

[Ope09] OPENSSL PROJECT: *OpenSSL: The Open Source toolkit for SSL/TLS*. http://www.openssl.org/, 2009. zuletzt abgerufen am 06.06.2010.

[P. 09] P. T. MAGAZIN: *Cloud Computing und die neuen Maßstäbe der Google-Ära*. http://www.pt-magazin.de/newsartikel/datum/2009/02/09/cloud-computing-und-die-neuen-massstaebe-der-google-aera/, Februar 2009. zuletzt abgerufen am 17.06.2010.

[Pat05] PATERSON, JIM: *Prevalence: Transparent, Fault-Tolerant Object Persistence*. http://onjava.com/pub/a/onjava/2005/06/08/prevayler.html, Juni 2005. zuletzt abgerufen am 12.06.2010.

[Pro] PROJECT VOLDEMORT: *Project Voldemort - A distributed database*. http://project-voldemort.com/. zuletzt abgerufen am 06.07.2010.

[Pro10] PRONSCHINSKE, MITCHELL: *CouchDB: Making it Okay to Work Offline*. http://architects.dzone.com/articles/couchdb-making-it-okay-work, Mai 2010. zuletzt abgerufen am 10.06.2010.

[Pyt10] PYTHON SOFTWARE FOUNDATION: *Python Programming Language*. http://www.python.org/, 2010. zuletzt abgerufen am 06.06.2010.

[Qui09] QUIRKEY, AAARON: *Sammy.js, CouchDB, and the new web architecture*. http://www.quirkey.com/blog/2009/09/15/sammy-js-couchdb-and-the-new-web-architecture/, September 2009. zuletzt abgerufen am 05.06.2010.

[Qui10a] QUIRKEY, AAARON: *Sammy*. http://code.quirkey.com/sammy/, 2010. zuletzt abgerufen am 05.06.2010.

[Qui10b] QUIRKEY, AAARON: *Sammy Documentation: Plugins*. http://code.quirkey.com/sammy/docs/plugins.html, 2010. zuletzt abgerufen am 24.06.2010.

[Sch06] SCHMIDT, ERIC: *Conversation with Eric Schmidt hosted by Danny Sullivan*. http://www.google.com/press/podium/ses2006.html, August 2006. zuletzt abgerufen am 19.06.2010.

[Sch08] SCHUSTER, WERNER: *Damien Katz Relaxing on CouchDB*. http://www.infoq.com/interviews/CouchDB-Damien-Katz, November 2008. zuletzt abgerufen am 16.06.2010.

[Sla10] Slater, Noah: *[VOTE] Apache CouchDB 1.0.0 release, first round*. http://couchdb-development.1959287.n2.nabble.com/VOTE-Apache-CouchDB-1-0-0-release-first-round-td5267885.html#a5267885, Juli 2010. zuletzt abgerufen am 09.07.2010.

[Smi09] Smith, David: *Browser support for CSS3 and HTML5*. http://www.deepbluesky.com/blog/-/browser-support-for-css3-and-html5_72/, November 2009. zuletzt abgerufen am 06.06.2010.

[Spr10] Sprout Systems Inc. Contributors: *What is Sproutcore?* http://www.sproutcore.com/what-is-sproutcore/, 2010. zuletzt abgerufen am 15.06.2010.

[Str98] Strozzi, Carlo: *NoSQL: a non-SQL RDBMS*. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page, 1998. zuletzt abgerufen am 11.06.2010.

[The10] TheCodingMonkeys: *SubEthaEdit*. http://www.subethaedit.de/index.de.html, 2010. zuletzt abgerufen am 16.06.2010.

[VG10] Van Gurp, Jilles: *CouchDB*. http://www.jillesvangurp.com/2010/01/15/couchdb/, Januar 2010. zuletzt abgerufen am 18.06.2010.

[Vog07] Vogel, Werner: *All Things Distributed: Amazon's Dynamo*. http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html, Oktober 2007. zuletzt abgerufen am 06.07.2010.

[W3C05] W3C: *Document Object Model (DOM)*. http://www.w3.org/DOM/, Januar 2005. zuletzt abgerufen am 09.07.2010.

[W3C09] W3C: *CGI: Common Gateway Interface*. http://www.w3.org/CGI/, Mai 2009. zuletzt abgerufen am 21.05.2010.

[Wei08] Weirich, Jim: *RAKE — Ruby Make*. http://rake.rubyforge.org/, 2008. zuletzt abgerufen am 09.06.2010.

[Wik10a] Wikipedia: *Gliederungseditor*. http://de.wikipedia.org/w/index.php?title=Gliederungseditor&oldid=74405760, 2010. zuletzt abgerufen am 02.06.2010.

[Wik10b] Wikipedia: *List of unit testing frameworks - Javascript*. http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#JavaScript, 2010. zuletzt abgerufen am 06.06.2010.

[Wik10c] Wikipedia: *Outliner*. http://en.wikipedia.org/w/index.php?title=Outliner&oldid=367410701, 2010. zuletzt abgerufen am 15.06.2010.

Abbildungsverzeichnis

Verzeichnis der Quelltextauszüge