Endbericht

PG 570: „Intercloud“

30. September 2013

technische universität dortmund

Fakultät für Informatik

Teilnehmer

Miguel Cifuentes Perelló, Fabian Coerschulte, Sebastian Fechner, Dominik Mohr, David Schoen, Tilo Schulz, Dominik Sparer, Sebastian Venier, Jaouad Zarouali, Niklas Zbick

Betreuer

Dipl.-Inf. Markus Doedt
Dipl.-Inf. Maik Merten
Dipl.-Inf. Hasan Ibne Akram
Dipl.-Inf. Johannes Neubauer
Dipl.-Inf. Michael Lybecait

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Abstract

Der Einsatz von Cloud-Computing-Technologien ermöglicht den Nutzern einen flexiblen Einsatz von Ressourcen, die nach Bedarf bereit gestellt werden können. Ein noch bestehendes Problem hierbei ist, dass es noch keine Standardisierung der, von den Cloud-Anbietern angebotenen, Schnittstellen gibt. Diese Tatsache führt zum so genannten Vendor-Lock-in-Effekt, was bedeutet, dass erstellte Software an einen Anbieter gebunden wird. Dies hat zur Folge, dass bei einem Anbieterwechsel Kosten für das Anbinden an den neuen Anbieter entstehen. In dieser Projektgruppe wird eine Zwischenschicht entwickelt, die die Schnittstellen der Cloud-Anbieter abstrahiert, um somit einen einfachen Wechsel des Anbieters zu ermöglichen. Dabei werden Strategien entwickelt, die es ermöglichen Produkte von verschiedenen Cloud-Anbietern für eine Webapplikation zu nutzen. Außerdem wird evaluiert, inwieweit einzelne Komponenten während der produktiven Nutzung der Webapplikation portiert werden können.

2. Einleitung

In diesem Kapitel erfolgt eine kurze Begründung des Projektthemas. Dabei soll erläutert werden, wo die Probleme bei der Nutzung von Cloud-Technologie auftreten und wie eine Lösung gefunden werden soll.

2.1. Motivation

Der Hype um das Thema Cloud-Computing und die damit verbundene Entwicklung, die von an Desktopcomputer gebundenen Applikationen weg zu Webanwendungen führt, scheinen unaufhaltsam. Immer mehr Anwender entdecken die Vorteile, die sich ihnen durch die Cloud bieten.
Ein weiterer sich abzeichnender Trend ist, dass viele kleine und mittlere Unternehmen mittlerweile auf eigene Rechenzentren, die damit verbundenen Kosten und möglicherweise vorkommende Probleme, verzichten und ihre Server in die Cloud auslagern [45]. Die vom Anwender benötigten Speicher- und Rechenkapazitäten werden nach Bedarf bei den Cloud-Anbietern eingekauft. Vor allem die Punkte Ausfallsicherheit, Wartung und nahezu unerschöpfliche, sofort nutzbare Ressourcen sind die ausschlaggebenden Argumente, um den Schritt zur Cloud zu gehen.

Bei dieser Überlegung werden allerdings oft die negativen Aspekte außer Acht gelassen. Einer dieser Aspekte ist die Abhängigkeit, in die man sich durch die fehlende Vereinheitlichung der Schnittstellen bei den großen Cloud-Anbietern begibt. Diese Abhängigkeit beim Cloud-Computing wird im Allgemeinen als Vendor Lock-In Problem bezeichnet [49]. Vor allem im Bereich der Datenspeicherung wird dies deutlich. Es werden zwar Schnittstellen zur Verfügung gestellt, die Anpassung der eigenen Software an diese Schnittstellen ist aber meist aufwendig und kostenintensiv. Somit wird auch ein möglicherweise in Betracht gezogener Wechsel des Anbieters erschwert. Aus diesem Grund kann bei einer auf eine Schnittstelle ausgerichteten Anwendung die sich ständig ändernde Marktsituation unter den Cloud-Anbietern nicht genutzt werden.
Die Aufgabe der Projektgruppe ist es, zu prüfen, inwieweit ein Ausweg aus dieser Abhängigkeit bei der Nutzung von Cloud-Technologien wegbar ist oder ermöglicht werden kann.

2.2. Aufbau des Dokuments

Zu Beginn werden im Kapitel 3 die Begriffe erklärt, die im Dokument wiederkehrend verwendet werden.

Daran anschließend werden die Rahmeninformationen zum organisatorischen Aufbau der Projektgruppe in Kapitel 4 gegeben.

Danach folgt der Ablauf des Projektes, beginnend mit der Seminarphase und Ex-

ploration (Kapitel 5), die sich mit der Analyse der PaaS-Anbieter beschäftigt.

Nachdem geeignete PaaS-Anbieter gefunden wurden, wird im Kapitel 6 die Suche auf Storage-Anbieter ausgeweitet und eine einheitliche Schnittstelle für diese entwickelt.

Als internes Demonstrationsprojekt wählte das Projektteam eine Applikation mit dem Titel „Content-Battle“. Dieses wird im Kapitel 7 noch genauer erläutert.

Der Installer (Kapitel 8) ist in der Lage, ein beliebiges WAR-Archiv auf der gewählten PaaS-Plattform zu deployen und mit einem Storage-Anbieter, der in der Storage-API implementiert wurde, zu verbinden.

In Kapitel 9 werden externe Projekte vorgestellt. Diese wurden dazu verwendet, zu überprüfen welcher Aufwand benötigt wird, die erstellte Storage-API, in fremde Applikationen einzubinden.

Abschließend wird in einem kurzen Fazit (Kapitel 10) festgehalten, welche Erkenntnisse wir während der Projektgruppe gewonnen haben und welche weiteren Vorhaben möglich oder sinnvoll wären.

3. Grundlagen

In diesem Kapitel werden die Begriffe erläutert, die Vorraussetzung für das Verständnis dieses Dokuments sind. Es wird zwischen den allgemeingültigen und den projektspezifischen Begriffen unterschieden.

3.1. Vorausgesetzte Kenntnisse

Dieses Dokument adressiert in erster Linie einen Leserkreis, der über grundlegende Informatikkenntnisse verfügt. Aus diesem Grund werden Begriffe und Akronyme als bekannt vorausgesetzt, die im alltäglichen Sprachgebrauch der Fachrichtung Informatik fest verankert sind. Dazu gehören beispielsweise Ausdrücke wie „HTML“ oder „Browser“. Alle weiteren Begriffe, die in diesem Dokument meist wiederkehrend verwendet werden, werden im Folgenden erklärt.

3.2. Grundlegende Begriffe

Zunächst werden Grundbegriffe vorgestellt. Diese sind allgemein bekannt und anerkannt. Da einige Begriffe wie „Cloud-Computing“ jedoch häufig in verschiedenen Zusammenhängen mit unterschiedlicher Bedeutung genannt werden, soll nun eine kurze Definition im Kontext dieser Projektgruppe erfolgen.

3.2.1. Cloud-Computing

Cloud-Computing ist eine Form der Bereitstellung von gemeinsam nutzbaren und flexibel skalierbaren IT-Leistungen durch nicht fest zugeordnete IT-Ressourcen über Netze. Typische Merkmale sind die Bereitstellung in Echtzeit auf Basis von Internet-Technologien und die Abrechnung nach Nutzung. Die IT-Leistungen können sich auf Anwendungen, Plattformen für Anwendungsentwicklungen und -betrieb sowie die Basisinfrastruktur beziehen und werden meist als Selbstbedienungstechnologie angeboten [44].

3.2.2. Service-Ebenen (IaaS, PaaS, SaaS)

IaaS Infrastructure as a Service (IaaS) ist im Rahmen von Cloud-Computing die Bereitstellung einer skalierbaren IT-Infrastruktur auf nicht eindeutig zugeordneten IT-Ressourcen über Netzwerke. Dieses Geschäftsmodell sieht eine Nutzung von Rechnerinfrastruktur nach Bedarf vor und bildet einen Gegenentwurf zum klassischen Erwerb. Die IT-Leistungen der Basisinfrastruktur stellen das Tätigkeitsfeld der Spezialisten für den IT-Betrieb sowie der IT-Dienstleister dar. Auf technologischer Ebene wird hier im Wesentlichen Rechen- und Speicherleistung auf virtualisierten Servern sowie Netzwerkinfrastruktur-Funktionalität mit hohem Standardisierungsgrad und intelligentem System-Management als Service bereitgestellt.

PaaS Platform as a Service (PaaS) ist im Rahmen von Cloud-Computing die Bereitstellung von gemeinsam nutzbaren Laufzeit- oder Entwicklungsplattformen auf nicht eindeutig zugeordneten IT-Ressourcen über Netzwerke. Dieses Geschäftsmodell stellt eine integrierte Laufzeit- und ggf. auch Entwicklungsumgebung als Dienst zur Verfügung, der dem Anwender gegenüber nach Nutzung in Rechnung gestellt wird. Mit den Cloud-Services der Ebene PaaS befassen sich System-Architekten und Anwendungsentwickler. PaaS beschreibt Services auf der Anwendungs-Infrastruktur-Ebene (Datenbanken, Integration und Security), die auf Basis von technischen Frameworks angeboten werden. Mit ihnen lassen sich Anwendungskomponenten entwickeln und plattformübergreifend integrieren.

SaaS Software as a Service (SaaS) ist im Rahmen von Cloud-Computing die Bereitstellung von gemeinsam nutzbarer Software auf nicht eindeutig zugeordneten IT-Ressourcen über Netzwerke. Unter SaaS versteht man ein Geschäftsmodell mit der Philosophie, Software als laufende Leistung basierend auf Internet-Techniken bereitzustellen, zu betreuen und zu betreiben, die in der Regel pro Aufruf abgerechnet wird, und die Software nicht länger als Lizenz an einen Nutzer zu verkaufen. SaaS richtet sich an Anwender. Geschäftsanwendungen werden als standardisierte Services von einem Dienstleister bereitgestellt. Dabei sind ihre Anpassungs- und Integrationsmöglichkeiten oft eingeschränkt. Desktop-, Kollaborations- und Kommunikations-Anwendungen sowie industriespezifische Geschäftsabläufe, die vollständig von der Technologie abstrahiert sind, fallen in diesen Bereich [44].

3.2.3. Cloud Storage

Unter einem Cloud Storage System versteht man einen Verbund von verteilten Datencentern, die dem Benutzer eine API (application programmable interface, siehe Abschnitt 3.2.12) zum Speichern von Daten zur Verfügung stellen. Die transparente Schnittstelle stellt dem Nutzer die Daten unabhängig der geografischen Verteilung auf unterschiedlichen Servern logisch zusammenhängend zur Verfügung. Cloud Storage kann je nach Anbieter aufgrund von redundanter und verteilter Datenhaltung eine hohe Ausfallsicherheit bieten. Aus diesem Grund ist es eine Alternative zu klassischen Backup-Maßnahmen. Die Kollaborations- und Versionierungsfunktionen, die von einigen Anbietern zur Verfügung gestellt werden, sind weitere Abgrenzungsmerkmale im Vergleich zur klassischen Datenhaltung [67].

3.2.4. Frontend-Backend-Architektur

Ein grundlegendes Konzeptionsmodell bei der Software-Entwicklung im Rahmen der PG ist die Frontend/Backend-Architektur gewesen. Dabei haben wir „Frontend“ als die Systemkomponente verstanden, die näher am Benutzer ist. Das sind in der Regel die bedienbaren Oberflächen. Im konkreten Fall sind es beispielsweise die dargestellten Webseiten. Wir haben uns bewusst für ein Frontend-Rendering entschieden, was zur Folge hatte, dass alle Elemente der Anzeige und Benutzerführung über den Client

(Browser) gesteuert wurden. Alternativ hätte der strukturelle Aufbau der Seiten und ihrer Inhalte (Rendering) auch direkt durch die Komponente erfolgen können, die näher am System ist (Backend). In diesen Bereich haben wir die Rechtevergabe und die funktionalen Dienste eingeordnet. Der Vorteil bei dieser Vorgehensweise ist bei Webseiten, dass nur die Inhalte nachgeladen werden müssen, die sich ändern. Dadurch können Webseiten unmittelbar auf Eingaben reagieren, ohne dass ein komplettes Neuladen der Seite erforderlich ist. Die Kommunikation zwischen Frontend und Backend kann auf unterschiedliche Art und Weise erfolgen. Im Rahmen der PG erfolgt die Kommunikation über sogenannte „RESTful-Service“.

3.2.5. RESTful-Service

Beim RESTful-Service handelt es sich um einen Webservice. Webservices sind dadurch gekennzeichnet, dass sie Programme, die in der Regel verteilt auf Netzwerkrechnern laufen, über das Internet miteinander verknüpfen. Der Architekturstil REST ist gekennzeichnet durch die Einhaltung folgender Prinzipien, bei denen die Art der Implementierung nicht vorgegeben ist.

Adressierbarkeit durch URI (Uniform Resource Identifier) RESTful-Services stellen für die Interaktion mit den Clients eine Auswahl an eindeutig adressierbaren Ressourcen (z.B. Bilder, Texte) und Diensten („Füge Bild hinzu“) zur Verfügung.

Einheitliche Schnittstellenoperationen Ressourcen werden mit Hilfe klar eingegrenzter Operationen (HTTP Verben) manipuliert (PUT, GET, POST, DELETE). PUT erzeugt neue Ressourcen, die durch DELETE gelöscht werden können. GET liefert den aktuellen Stand einer Ressource. POST ändert den Status einer Ressource und überträgt ggf. weitere Teilinformationen.

Selbstbeschreibende Repräsentationsformen Die unter einer Adresse zugänglichen Dienste können unterschiedliche Darstellungsformen (Repräsentationen) haben. Ein REST-konformer Server kann je nachdem, was die Anwendung anfordert, verschiedene Repräsentationen ausliefern (z. B. HTML, JSON oder XML). Damit kann der Standard-Webbrowser über die gleiche URI-Adresse sowohl den Dienst, als auch die Beschreibung oder Dokumentation abrufen.

Zustandslosigkeit Jede Interaktion mit Ressourcen ist zustandslos, d.h. es werden genau die Informationen übermittelt, die für die Ausführung des Arbeitsauftrages benötigt werden. Zwischenzustände (wie z.B. nur teilweise auf den Server geladene Dateien) sind aus diesem Grund nicht abrufbar. Diese Architektur bietet den Vorteil, dass auf Seiten des Servers keine Sitzungsverwaltung erforderlich ist, da jede Anfrage in sich abgeschlossen ist [61].

3.2.6. JSON

Bei JSON (JavaScript Object Notation) handelt es sich um ein menschenlesbares Datenformat für die Repräsentation hierarchischer Datensätze und für den Datenaustausch im Umfeld von JavaScript. Im Vergleich zu XML-Nachrichten enthalten JSON-Daten weniger Rahmeninformationen. Dies sorgt bei gleichem Inhalt für eine Reduzierung der Dateigröße. Außerdem können die im JSON-Format versendeten Nachrichten ohne Zwischenschritte direkt in JavaScript-Objekte konvertiert und weiterverwendet werden [41].

3.2.7. SOAP

SOAP ist ein Protokoll zum zustandslosen Austausch von XML-Nachrichten über ein Computernetzwerk. Es stellt Regeln für das Nachrichtendesign auf und regelt, wie Daten in der Nachricht abzubilden und zu interpretieren sind. Außerdem ist eine Konvention für entfernte Prozeduraufrufe mittels SOAP-Nachrichten vorgegeben. SOAP stützt sich auf XML zur Repräsentation der Daten und auf Internet-Protokolle der Transport- und Anwendungsschicht (vgl. TCP/IP-Referenzmodell) zur Übertragung der Nachrichten [64].

3.2.8. IDE

Eine IDE (integrated development environment) ist eine Entwicklungsumgebung, die eine Sammlung von Anwendungen zur effizienten Bearbeitung einzelner Aspekte der Softwareentwicklung zur Verfügung stellt. Beispielsweise sind dies Komponenten wie Texteditoren, Compiler und Debugger. Umfangreichere IDEs können u.a. auch Werkzeuge für Modellierungs- und Planungsaufgaben beinhalten.

3.2.9. Entwicklungsumgebung Eclipse

Eclipse ist eine quelloffene IDE zur Softwareentwicklung. Ursprünglich wurde Eclipse für die Programmiersprache Java konzipiert, aber mittlerweile wird es wegen seiner Erweiterbarkeit auch für viele andere Entwicklungsaufgaben eingesetzt.

3.2.10. Apache Maven

Apache Maven ist ein Werkzeug zur Definition aller für einen Build-Prozess notwendigen Elemente. In der zentralen Datei `pom.xml` (Project Object Model) werden die benötigten Ressourcen definiert. Beim Build-Prozess mit Hilfe von Maven werden nicht nur Paketierungsinformationen festgelegt, sondern auch automatisch die aktuellen bzw. die genau festgelegten Versionen der referenzierten Java-Bibliotheken aus einem zentralen Repository im Internet geladen [43].

3.2.11. Git

Bei Git handelt es sich um ein Versionierungswerkzeug, das auf einem dezentralen, verteilten System beruht. Dabei werden von einem System zum anderen nicht nur die Sourcen, sondern auch die die komplette Historie übertragen. Es gibt im Gegensatz zu anderen Versionierungswerkzeugen keinen zentralen Server. Anbieter von PaaS-Diensten nutzen Git als Werkzeug für die Übertragung der Daten vom lokalen Dateisystem auf das System in der Cloud. Durch Versionsverwaltungssoftware kann u.a. Software-Entwicklung in Kollaboration mit anderen erfolgen.

3.2.12. API

Eine Programmierschnittstelle (englisch application programming interface, API; Schnittstelle zur Anwendungsprogrammierung) ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird. So bietet die API von Cloud Storages einen Zugang zu den Dateien ohne beispielsweise die anbietereigene Webseite nutzen zu müssen.

3.2.13. Java Archive (JAR-Datei)

Ein Java Archive (JAR-Datei) ist eine ZIP-Datei, die zusätzliche Metadaten in einer Datei `META-INF/MANIFEST.MF` enthalten kann. JARs werden vor allem zur Verteilung von Java-Klassenbibliotheken und -Programmen eingesetzt [72].

3.2.14. Webcontainer

Webcontainer sind Bestandteil der Spezifikation der Java Enterprise Edition, die eine Software-Architektur für transaktionsorientierte Java-Anwendungen - und dabei insbesondere Web-Anwendungen - festlegt. Die Spezifikation unterscheidet diverse logische Einheiten, sog. Container. Auch der Webcontainer ist eine solche logische Einheit, und stellt die Laufzeitumgebung für Servlets und Java Server Pages (JSP) zur Verfügung [53]. Verschiedene Webserver wie Apache Tomcat, Jetty oder Glasfish stellen gemäß der JEE-Spezifikation entsprechende Webcontainer zur Verfügung. Den Vorgang des Einstellens einer Anwendung auf dem Webcontainer bis hin zu seiner eigentlichen Ausführung nennen wir im Folgenden Deployment. Angelehnt an diesen Begriff verwenden wir das Verb „deployen“.

3.2.15. Web Archive (WAR-Datei)

Web Archive oder Web Application Archive ist ein Dateiformat, das beschreibt, wie eine vollständige Webanwendung nach der Java-Servlet-Spezifikation in eine Datei im JAR- bzw. ZIP-Format verpackt wird. Solche Dateien haben immer die Endung `.war` und werden daher umgangssprachlich auch „WAR-Datei“ genannt [73]. Zahlreiche PaaS-Anbieter, die einen Webserver zur Verfügung stellen, erwarten dieses Dateiformat, um die Webanwendung deployen zu können.

3.3. Projektspezifische Begriffe

Im Laufe des Projektes zeigte sich, dass einzelne Aufgaben bzw. Softwarekomponenten mit eigenen Begriffen belegt wurden, damit eine klare Abgrenzung möglich wurde und in Diskussionen Mehrdeutigkeiten reduziert wurden. Deshalb wurden folgende Begriffe zu eindeutigen Synonymen, die Aufgabenbereiche und Teilkomponenten in Kurzform eindeutig beschreiben.

3.3.1. Storage API

Unter dem Begriff „Storage API“ ist das in Java implementierte Interface zu verstehen, das eine einheitliche Steuerung unterschiedlicher Cloud Storages ermöglicht. So wird beispielsweise die Funktion „Datei-Hochladen“ je nach API des Zielsystems übersetzt und ausgeführt. Idealerweise müssen die Nutzer des von dieser PG entwickelten Interfaces die unterschiedlichen Spezifikationen der Storages-Schnittstellen nicht weiter berücksichtigen.

3.3.2. Content-Battle

Unter „Content-Battle“ ist die konkrete Umsetzung des Demonstrationsprojekts zu verstehen. Im Sinne der PG handelt es sich dabei um eine Web-anwendung, die dadurch gekennzeichnet ist, zahlreiche Ressourcen wie z.B. Bilder vorhalten zu müssen. Weitere Informationen sind dem Kapitel 7 zu entnehmen.

4. Organisation der Projektgruppe

Dieses Kapitel gibt einen Einblick in die Arbeitsweise und Fortschritte der Projektgruppe. Als Erstes ist beschrieben, wieso wir einen Projektleiter brauchten und welche Aufgaben dieser hatte. Im Anschluss ist der Ablauf der wöchentlichen Gruppentreffen dargelegt. Der nächste Abschnitt dieses Kapitels stellt die Anforderungen an ein großes Projekt dar und erläutert, welche Lösungen es für diese Anforderungen gibt und für welche expliziten Lösungen sich die Projektgruppe entschied. Die letzten zwei Abschnitte beschäftigen sich mit den für das erste Semester gesetzten Milestones und der Einteilung der Mitglieder der Projektgruppe in ihre einzelnen Arbeitsgruppen.

4.1. Projektleitung

Für viele Studenten ist die Projektgruppe das erste große Software Projekt, an dem sie arbeiten. Sie arbeiten zwar im Zuge des Software Praktikums an kleineren Projekten, aber dort werden ihnen der Ablauf und die Deadlines vorgegeben. Außerdem sind die Fähigkeiten und Erfahrungen der Studenten auf unterschiedlichem Niveau und am Anfang der Projektgruppe den anderen noch nicht bekannt gewesen. Sebastian Venier wurde zum Projektleiter und David Schoen zu seinem Stellvertreter gewählt.

Ein Teil der Aufgaben der Projektleitung war es, die wöchentlichen Sitzungen zu leiten, zu planen und zu moderieren. Ein weiterer Teil bestand darin, die Leistung und Erfahrung der einzelnen Gruppenmitglieder schnell zu beurteilen und Vorschläge für die Aufteilung in die Arbeitsgruppen zu machen. Diese Vorschläge wurden dann in der Gruppe diskutiert. Ein weiterer administrativer Teil war es, Übersicht über den Fortschritt der einzelnen Arbeitsgruppen zu behalten und zu intervenieren, falls Deadlines überschritten zu werden drohten. Dazu war nötig, dass sich die Projektleitung Überblick über alle Teilgebiete aneignete. Die Deadlines wurden jedoch nicht von der Projektleitung, sondern im großen Plenum auf dem wöchentlichen Treffen unter Gleichberechtigung aller Team-Mitglieder bestimmt. Schließlich war es ebenfalls Aufgabe der Projektleitung, die Protokolle zu kontrollieren und wenn nötig zu ergänzen.

4.2. Gruppentreffen

Der Projektleiter und sein Stellvertreter waren für die allgemeine Organisation und die Leitung der wöchentlichen Treffen verantwortlich. Die Ergebnisse dieser Treffen wurden durch einen Protokollanten festgehalten, welcher wöchentlich wechselte. Die Treffen dauerten jeweils ca. 2 Stunden. Die einzelnen Arbeitsgruppen innerhalb der PG trafen sich wenn nötig zusätzlich. Die Treffen wurden dafür genutzt, die Fortschritte der Teilaufgaben zu präsentieren, aufgetretene Fragen zu klären, das weitere Vorgehen zu planen und Aufgaben zu verteilen. Die Plenums-Treffen wurden von der Projektleitung geplant und geleitet. Feste Bestandteile waren Berichte der

PG-Mitglieder, Präsentationen der Ergebnisse sowie die Bestimmung des nächsten Protokollanten. Die einzelnen Arbeitsgruppen organisierten sich selbständig.

4.3. Anforderungen eines großen Software Projektes

Große Software Projekte erfordern, dass mehrere Personen zusammen arbeiten. Im Zuge dessen kann es schwierig sein, einen Überblick über den Fortschritt zu behalten. Die menschlichen Ressourcen müssen effektiv eingesetzt werden. Alle Personen müssen den gleichen Stand in ihrer Software haben. Bei der Entwicklung von Software können Fehler entstehen, welche dokumentiert werden müssen, um sie später zu beheben. Großprojekte werden oftmals in kleinere überschaubare Projekte unterteilt. Die einzelnen Teilprojekte müssen am Ende wieder zu einem Ganzen zusammengefügt werden. Dies alles erfordert organisatorischen Aufwand. Dementsprechend ist es nötig, Werkzeuge zu haben, mit denen solche Probleme gelöst werden können.

4.3.1. Projektmanagement-Werkzeuge

Einige dieser Probleme lassen sich durch Projektmanagement-Werkzeuge in Angriff nehmen. Diese enthalten in den meisten Fällen einen chronologischen Verlauf, welcher allen Benutzern erlaubt, zu sehen, was geschehen ist. Das Ticketsystem eines Projektmanagement-Werkzeugs eignet sich Aufgaben zu verteilen, Fehler festzuhalten und Milestones zu setzen. In vielen existierenden Projektmanagement-Softwarelösungen ist außerdem ein Wiki enthalten, welches für viele organisatorische Aufgaben, wie Protokolle von Treffen und weiteren Informationsaustausch, genutzt werden kann. Die Projektgruppe entschied sich für die Nutzung von Redmine [63] als Projektmanagement-Lösung, da einige PG-Mitglieder bereits zuvor damit gearbeitet hatten und die Versionsverwaltung einfach zu integrieren ist. Das Redmine wurde vom ITMC der TU-Dortmund zu Verfügung gestellt.

4.3.2. Versionsverwaltung

Um die Software bei allen Beteiligten auf dem gleichen Stand zu halten eignet sich eine Versionsverwaltungssoftware. Diese Tools protokollieren alle Änderungen und halten fest, wer diese gemacht hat. Es ist möglich, zu jeder alten Version zurückzukehren, da alle Änderungen archiviert werden. Ein weiteres Feature ist das Erstellen von sogenannten Branches, mit denen verschiedene Lösungsansätze gleichzeitig bearbeitet werden können. Es gibt viele existierende Lösungen mit unterschiedlichen Vor- und Nachteilen. Die Bekanntesten dieser Softwarelösungen sind Git, Subversion, Mercurial und Darcs [50, 69, 58, 47]. Die Projektgruppe entschied sich für Subversion, da dieses zentral verwaltet ist und im Voraus schon von den meisten Gruppenmitgliedern genutzt wurde. Es konnte einfach in das Redmine integriert werden und wurde ebenfalls vom ITMC der TU Dortmund bereitgestellt.

4.3.3. Build Management

In Projekten einer gewissen Größe werden häufig externe Bibliotheken genutzt. Außerdem müssen Teilprojekte miteinander arbeiten. Build-Management-Tools erleichtern diese Arbeiten. Sie übernehmen das Verlinken der Bibliotheken und bestimmen die Reihenfolge, in der kompiliert wird. Sie übernehmen das Testen und Verteilen der Software. Dies läuft automatisch ab, so dass sich der Programmierer auf die wesentlichen Arbeiten konzentrieren kann. Außerdem will man häufig nicht den gesamten Sourcecode neu kompilieren. Build-Management-Tools haben Verfahren, mit denen nur bestimmte Teile neu gebaut werden müssen. Da wir mit Java [60] arbeiteten, waren die zur Verfügung stehenden Softwarelösungen Apache Ant, Apache Maven und Gradle [42, 57, 52]. Maven war nach Ansicht der Projektgruppe die beste Lösung. Es ist schlank, weit verbreitet und vergleichbar leicht zu konfigurieren, in Java geschrieben und dient nicht nur zum bloßen Kompilieren von Projekten, sondern auch zum Einbinden von Programmbibliotheken über ein zentrales Archiv (sowohl lokal als auch im Internet verfügbar). Darüber hinaus ist Maven über Plugins erweiterbar und damit sehr mächtig. So lassen sich beispielsweise direkt WAR-Archive erstellen oder Anwendungen über ein Webcontainer-Plugin mit einem einzigen Befehl in z.B. einem eingebetteten Tomcat Webserver starten. Aufgrund dieser Vorteile entschieden wir uns für den Gebrauch dieser Software.

4.3.4. LaTeX

LaTeX [70] ist ein Textsatz-Werkzeug, das auf wissenschaftliche Arbeiten und Berichte ausgelegt und von uns genutzt wurde. Da einzelne Dokumente in LaTeX eingebunden werden können, konnte so das Verfassen einzelner Abschnitte aufgeteilt werden. LaTeX arbeitet nicht nach dem WYSIWYG-Prinzip[1]. Die endgültige Formatierung sieht der Autor noch nicht während des Schreibens. Spezielle Formatierungen, wie z.B. Überschriften, Absätze etc. können durch Befehle gekennzeichnet werden. Das Layout von LaTeX gilt dabei als sehr sauber. Eine einfache Speicherung in anderen Formate wie PDF ist auch möglich.

4.4. Milestones

Die beiden Semester in dem die Projektgruppe statt fand enthielten einige Milestones, welche zum Teil durch die Betreuer und zum anderen von uns bestimmt worden sind. In Tabelle 1 sind diese zusammen mit ihren Deadlines und ihrem Fertigstellungszeitpunkt angegeben.

4.5. Verantwortlichkeiten

Zu Beginn jedes Milestones wurden die Aufgaben an einzelne PG-Mitglieder durch die gesamte Gruppe verteilt. In Tabelle 2 ist nachzulesen, wie die einzelnen Arbeits-

[1] What you see is what you get

Milestone	**Deadline**	**Fertigstellungszeitpunkt**
Seminarausarbeitung und Präsentation	18.10.2012	18.10.2012 2 nicht abgegeben
Exploration der Anbieter	15.11.2012	15.11.2012
Implementierung Storage API	29.11.2012	29.11.2012
Testen der Storage API	23.12.2012	17.01.2013
Konzept Content Battle	20.12.2012	20.12.2012
Implementierung Backend	07.03.2013	07.03.2013
Testen des Backends	21.03.2013	21.03.2013
Implementierung Frontend	21.03.2013	16.05.2013
Erstellung Zwischenbericht	31.03.2013	17.04.2013
Abschluss Programmierarbeiten	04.07.2013	01.08.2013
Korrektur Zwischenbericht	11.07.2013	29.07.2013
Inhaltsvorstellung Endbericht	19.07.2013	19.07.2013
Erstfassung Endbericht	08.08.2013	29.08.2013
Zweitfassung Endbericht	05.09.2013	13.09.2013

Tabelle 1: Milestones des ersten Semesters

gruppen aufgeteilt waren.

Aufgaben	**Verantwortlicher**
Projektleitung	Sebastian Venier David Schoen
Implementierung Google Drive	Dominik Mohr Tilo Schulz Sebastian Venier
Implementierung 4Shared	Miguel Cifuentes Niklas Alexander Zbick David Schoen
Implementierung Mediafire	Jaouad Zarouali Sebastian Fechner Dominik Sparer
Impementierung Local	Fabian Coerschulte
Leitung des Testbetriebes	David Schoen
Tests Storage	Alle
Konzepterarbeitung Content-Battle	Miguel Cifuentes Sebastian Venier Niklas Alexander Zbick
Implementierung Backend	Tilo Schulz Dominik Mohr
Tests Backend	Sebastian Venier
Teamleitung Frontend	David Schoen
Implementierung und Tests des Frontends	David Schoen Fabian Coerschulte Jaouad Zarouali Miguel Cifuentes Niklas Alexander Zbick Sebastian Fechner
Verantwortlichkeit Zwischenbericht	Dominik Mohr Dominik Sparer
Implementation Installer Backend	Dominik Mohr Jaouad Zarouali Tilo Schulz
Implementation Installer Frontend	Sebastian Venier Fabian Coerschulte
Integration von Storage API in Third-Party Software	Niklas Alexander Zbick Sebastian Fechner Dominik Sparer
Optimierung Content-Battle	David Schoen
Verantwortlichkeit Endbericht	Miguel Cifuentes Tilo Schulz
Abschluss Präsentation	Miguel Cifuentes Sebastian Venier David Schoen

Tabelle 2: Aufgaben der einzelnen Personen

5. Seminarphase und Exploration

Um erste Wissensgrundlagen für spätere Entscheidungsprozesse zu schaffen, startete die PG Intercloud mit einer Seminarphase. In dieser wurden die Grundlagen für den Einstieg in den Themenkomplex Cloud-Dienste gelegt. Jeder einzelne wurde damit beauftragt, sich mit einem Anbieter auseinanderzusetzen und in einer Ausarbeitung die Erkenntnisse zusammenzufassen. Die Themenvergabe erfolgte durch die Betreuer. Die untersuchten Angebote, die kostenlos oder kostenpflichtig waren, konnten zumeist den zwei Kategorien PaaS und IaaS zugeordnet werden.

Die bei der PG-Vorstellung geäußerte Idee, eine Webseite aufzubauen, die sich in erster Linie dadurch auszeichnet, dass sie mehrere Fähigkeiten unterschiedlicher Cloud-Anbieter miteinander verbindet, spielte von Anfang an eine Rolle bei der Selektion der richtigen Angebote. So spielten beispielsweise Google Docs oder Google Calender als fertige Applikationen bei den Untersuchungen eine untergeordnete Rolle. Das Hauptaugenmerk lag auf Angeboten, die Datenspeicher (Storage Anbieter) und/oder Webservices zur Verfügung stellen. Alle Dienste wurden in diesem Zusammenhang auf Beschränkungen und Vorgaben untersucht, um beurteilen zu können, ob eine Integration des internen Testprojekts realisierbar wäre oder nicht.

Für die Einschätzung des jeweiligen Angebotes, fand ein grundsätzlicher Wissenstransfer zwischen den Gruppenmitgliedern, im Rahmen einer zweitägigen Präsentationsphase, statt. Diese Termine waren die Kick-Off-Veranstaltung für den Start der Projektphase, in der gemeinsam ein Konzept und eine Vorgehensweise erarbeitet wurde.

5.1. PaaS Anbieter

Zu Beginn der Projektgruppe sollte ein Überblick über die Anbieter und Möglichkeiten von PaaS geschaffen werden. Im Rahmen eines Seminars präsentierten die Teilnehmer eine Auswahl an Lösungen in diesem Bereich. Auf dieser Grundlage wurden dann im Anschluss einige Plattformen ausgewählt, die im weiteren Verlauf für das Projekt genutzt werden sollten. In diesem Abschnitt wird eine Auflistung der untersuchten Produkte vorgenommen. Dabei wird jedes lediglich kurz angerissen. Für eine ausführliche Beschreibung sei auf die einzelnen Ausarbeitungen aus der Seminarphase verwiesen. Die Diskussion der Anbieter erfolgt in Abschnitt 5.2.

5.1.1. Apache Deltacloud

Apache Deltacloud soll durch eine Schnittstelle unabhängig vom Anbieter diverse Lösungen im Bereich IaaS nutzbar machen [65]. Um Wartungsaufwand zu verringern, ist dabei stets Abwärtskompatibilität in der Deltacloud-REST-API gewährleistet. Nach Versionsänderungen in dieser API liefern Anfragen an die REST-Schnittstelle also weiterhin das gleiche Resultat da vorhandenen Funktionen lediglich neue optionale Parameter hinzugefügt werden.
Die Architektur der Apache Deltacloud lässt sich in Server inklusive REST-Schnitt-

stelle, Treiber für verschiedene IaaS-Anbieter und Clients für die Kommunikation mit dem Server unterteilen.
Durch den Server wird die Kommunikation mit den Diensten der Anbieter vollzogen und ermöglicht dem Entwickler über die Schnittstelle abstrahierten Zugriff. Ein Client ist eine Bibliothek für eine bestimmte Programmiersprache, die dem Entwickler direkte Interaktion mit dem Deltacloud-Server ermöglicht. Treiber ermöglichen die Nutzung von Diensten eines Anbieters und bilden somit das Bindeglied zwischen der Deltacloud und einer IaaS.
Der Server der Deltacloud selbst ist in der Programmiersprache Ruby implementiert. Für Anwender steht sowohl eine Sourcecode-Version als auch eine binäre Version zur Nutzung zur Verfügung. Die Installation des Servers lässt sich durch RubyGems mit zwei Kommandos erstellen. Dienste der Deltacloud lassen sich direkt nach dem Start des Servers mit einem übergebenen Treiber nutzen. Über einen Webservice des Servers lässt sich per HTTP das Web-Interface nutzen, über das eine benutzerfreundliche Verwaltung der aktuell zur Verfügung stehenden Dienste externer Anbieter möglich ist. Durch die REST-Schnittstelle des Servers beinhaltet jeder Request der Deltacloud eine Authentifizierung, die durch eine HTTP-Basic-Authentication realisiert wird.
Die Konsequenz für Anwender ist, dass ein Treiber existieren muss, der den gewünschten zu nutzenden Anbieter unterstützt. Aktuell werden Treiber für zwölf verschiedene Dienste angeboten.
Programmierer können statt der REST-Schnittstelle ebenfalls Client-Bibliotheken für Ruby oder C/C++ zur Kommunikation mit dem Deltacloud-Server nutzen.

Die Vorteile der Apache Deltacloud für die Projektgruppe sind neben der einfachen Handhabung in Bezug auf Installation und Start eines Servers auch die Abwärtskompatibilität der Schnittstelle. Ein Nachteil für die Projektgruppe besteht in der Tatsache, dass nur Client-Bibliotheken für die Programmiersprachen Ruby und C/C++ zur Verfügung stehen. Von vornherein lag der Fokus beim Projekt jedoch auf der Programmiersprache Java.

5.1.2. Apache Libcloud

Die Libcloud ist ein Top-Level-Projekt der Apache Software Foundation, das von dem Unternehmen Cloudbrick erstellt wurde [68]. Es besteht aus einer Bibliothek für die Programmiersprache Python und ermöglicht einen abstrahierten Zugriff auf APIs unterschiedlicher Cloud-Anbieter aus einer Applikation heraus. Das ermöglicht eine flexible Nutzung von verschiedenen Cloud-Services ohne aufwändiges Anpassen von eigener Software.

Bei der Apache Libcloud ist es für die Projektgruppe von Vorteil, dass die Kommunikation über eine REST-Schnittstelle realisiert ist. Der Nachteil ist neben der Realiserung der Libcloud in der Programmiersprache Python auch die Tatsache, dass die Libcloud durch synchrone Kommunikation nicht „thread safe“ ist, es also

bei gleichzeitiger Ausführung mehrerer Instanzen zu Fehlern kommen kann.

5.1.3. Amazon WebServices

Die Amazon WebServices (AWS) umfassen unter anderem Produkte zur Datenverarbeitung, Content Distribution, zu Datenbanken und Storage [46].
Jeder Dienst ist über APIs per REST, SOAP oder Query ansprechbar, daher ist man als Entwickler bei der Benutzung der Services nicht an eine bestimmte Programmiersprache gebunden. Das Produkt Elastic Beanstalk ermöglicht die gemeinsame Nutzung von Diensten der AWS als PaaS. Unterstützt werden Anwendungen in .Net, PHP oder Java. Um beispielsweise eine Java Anwendung zu starten, wird eine WAR-Datei zu den AWS geladen und dort auf einem Apache-Tomcat Server ausgeführt. Die Anwendung kann virtuelle Rechenressourcen (Produktname: EC2-Instanzen), Datenbanken oder weitere Dienste von Amazon nutzen. Der Entwickler kann die Flexibilität einer laufenden Anwendung durch Steuerung der Anzahl laufender Instanzen (AutoScaling), Verteilung des eingehenden Anwendungsverkehrs auf mehrere EC2-Instanzen (Elastic Load Balancing) und Überwachung von Ressourcen (CloudWatch) selbst bestimmen.

Die Vorteile der AWS für die Projektgruppe sind neben dem breit gefächerten Angebot an Leistungen auch die Flexibilität in der Steuerung laufender Instanzen und beanspruchter Ressourcen. Der Nachteil ist die Tatsache, dass für die Nutzung des Dienstes Daten einer Kreditkarte angegeben werden müssen, auch wenn nur zu Testzwecken kostenlose Kontingente in Anspruch genommen werden.

5.1.4. CloudBees

CloudBees soll den Softwarelebenszyklus komplett in die Cloud bringen und ist speziell für Java-Anwendungen gedacht [75]. Das Angebot ermöglicht einem Entwickler, aufbauend auf einer prinzipiell frei wählbaren Lösung aus dem Bereich IaaS, die Plattform von der Entwicklung von Software bis hin zum Ausführen der fertigen Programme zu nutzen. Per SaaS werden über Schnittstellen verschiedene Werkzeuge zum Erstellen von Code, Builds und Tests zur Verfügung gestellt. Darüber hinaus kann über CloudBees Software verwaltet werden (inkl. Deployment). Das Angebot soll die Java Plattform uneingeschränkt unterstützen und stellt auch Werkzeuge wie Maven, SVN oder Git bereit.

Bei CloudBees war es für die Projektgruppe von Vorteil, dass das Produkt konkret für die Entwicklung von Java-Anwendungen gedacht ist.

5.1.5. Force.com

Force.com ist ebenfalls eine Entwicklungs- und Ausführungs-Plattform für Webapplikationen [71]. Ein Nutzer hat die Möglichkeit, über eine IDE lokal oder aber über einen Webservice im Browser eigene Software zu entwickeln. Bei einer lokalen IDE

wird die Anbindung über ein Plugin von Force.com oder eine extra zugeschnittene Eclipse-Version erreicht. Wenn man sich für die Softwareentwicklung im Browser entscheidet, hat man die Möglichkeit, einen Großteil seiner Webapplikation ohne Schreiben von eigenem Code zu realisieren. Per Baukastensystem kann man sich kleinere Anwendungen direkt erstellen, man kann aber ebenfalls per HTML, Ajax, Javascript oder Apex Code einfügen oder verändern. Durch die Integrations-APIs kann die eigene Geschäftslogik in einer Programmiersprache der eigenen Wahl geschrieben werden und per SOAP, REST oder anderen zur Verfügung gestellten Methoden an die Services von Force.com angebunden werden.

Der Vorteil von Force.com für die Projektgruppe ist neben der ausführlichen Dokumentation und der Möglichkeit in Java zu entwickeln auch, dass kostenlos entwickelt werden kann. Der Nachteil ist die Tatsache, dass das Design der graphischen Oberfläche einer Anwendung nicht gänzlich frei ist, sondern stets aus Banner und grafischen Registrierkarten besteht.

5.1.6. Google Apps Application APIs

Die Google Apps Application APIs sind Programmier- und Kommunikationsschnittstellen für verschiedene von Google angebotene Applikationen [66]. Die Nutzung erlaubt es, die mit einem Google-Konto frei zugänglichen Applikationen in eigenen Anwendungen als Backend zu nutzen, ohne dass Nutzer im Frontend dieses bemerken. Nutzbar sind über die Schnittstellen Google Calendar, Google Tasks, GMail, Google Contacts, Google Drive und Google List. Zusätzlich bieten die Google Apps Application APIs Schnittstellen für die Authentifizierung von Benutzern und Administration von Google Domains mit deren Services. Um die angebotenen APIs zu kombinieren, kann Google Apps Script genutzt werden. Die Google Apps Application APIs werden als Bibliotheken für verschiedene Programmiersprachen wie z.B. Java, .Net, JavaScript oder PHP angeboten. Der Datentransfer wird per JSON durchgeführt. Mit Hilfe dieses Formats sind die API-Funktionen über REST ansprechbar.

Bei den Google Apps Application APIs ist es für die Projektgruppe von Vorteil, dass kostenlos in der Programmiersprache Java entwickelt werden kann. Der Nachteil ergibt sich aus den restriktiven Nutzungsbedingungen von Google. Die Restriktionen ergeben sich zum Beispiel aus den in den Nutzungsbedingungen enthaltenen Datenschutzbestimmungen, die Google die umfassende, dienstübergreifende Auswertung von Nutzerdaten erlauben. Die Zwecke dieser Auswertung bleiben dem Nutzer dabei verborgen und schränken seine Privatsphäre ein.

5.1.7. Google App Engine

Die Google App Engine ermöglicht dem Entwickler das Bereitstellen und Ausführen von eigenen Anwendungen auf der Infrastruktur von Google [48]. Unterstützt wer-

den dabei die Programmiersprachen Java, Python und Go. Anwendungen werden von der App Engine in einer Sandbox-Umgebung ausgeführt. Durch diesen Mechanismus wird die Skalierbarkeit und Sicherheit zur Laufzeit gewährleistet. Eine Besonderheit ist, dass Anwendungen durch die Sandbox einigen Restriktionen wie der maximalen Laufzeit oder dem verzögerten Ausführungsbeginn unterliegen. Die Entwicklung für die App Engine wird durch ein Software Development Kit (SDK) erleichtert. Per Kommandozeile oder IDE-Plugin lässt sich damit Software lokal unter Bedingungen der App Engine testen und schließlich auch ins tatsächliche System hochladen. Eine Verwaltungskonsole erlaubt aus einem Browser heraus die Verwaltung von Instanzen bezüglich der Skalierbarkeit, Zuweisung von Ressourcen oder auch Zugriffsberechtigungen. Bei der Java-Entwicklung werden zur Nutzung von Diensten Java Standardtechnologien wie Java Data Objects (JDO) zur persistenten Speicherung von Java-Objekten, Java Persistence API (JPA) als Schnittstelle für Zuordnung und Übertragung von Java-Objekten zu Datenbankeinträgen oder auch JavaMail genutzt.

Der Vorteil der Google App Engine für die Projektgruppe ist neben der Möglichkeit in Java zu entwickeln auch, dass der Dienst im kleinen Rahmen kostenlos nutzbar ist. Der Nachteil ist neben den restriktiven Nutzungsbedingungen im Hinblick auf den Datenschutz die Tatsache, dass durch die Sandbox zum Beispiel Grenzen in maximaler Ausführungsdauer einer Anwendung gegeben sind.

5.1.8. Heroku

Heroku ist ein weiterer Anbieter von PaaS und unterstützt Software, die in den Programmiersprachen Java, Ruby, Python, Lojure, Scala oder JavaScript geschrieben ist [59]. Das Deployment erfolgt mit dem Versionierungssystem Git. Einer Anwendung können durch einen Entwickler diverse sogenannte Dynos bereitgestellt werden. Das sind Instanzen, die als „WebDyno“ Anfragen per HTTP verarbeiten und entsprechenden Code ausführen oder als „Worker Dyno“ im Backend arbeiten und zum Beispiel zeitlich geplante Aufgaben erledigen. Über die Anzahl der Dynos lässt sich eine laufende Anwendung skalieren. Für die Entwicklung mit Heroku steht das „Heroku Toolbelt“ bereit. Dieses bietet einen Client, der über eine Kommandozeile Zugriff auf die Heroku Web API und somit auf die Verwaltung und Konfiguration von Anwendungen bei Heroku ermöglicht. Außerdem enthält es Git zur Bereitstellung von eigenen Anwendungen bei Heroku sowie Foreman zum lokalen Testen. Das Entwickeln des Codes ist problemlos in IDEs wie Eclipse möglich. Da Heroku auf der Infrastruktur von Amazon läuft, sind eine Reihe von Vorteilen bezüglich der Sicherheit von eigenen Daten gegeben.

Vorteil von Heroku ist für die Projektgruppe, dass kostenlos in der Programmiersprache Java entwickelt werden kann. Der Nachteil ergibt sich aus hohen Latenzzeiten für europäische Nutzer sowie dem Datenschutz im Hinblick auf die Nutzungsbedingungen.

5.1.9. OpenShift

OpenShift dient der Entwicklung, dem Testen und dem Betreiben von Applikationen innerhalb der Cloud und bietet dafür Dienste im Sinne von PaaS an [62]. Neben CPU-Ressourcen, Arbeitsspeicher und Festplattenspeicher stehen dem Nutzer ebenfalls Apache- und JBoss-Server (Anwendungsserver nach dem Java-EE-Standard) zur Verfügung. Bei OpenShift ist ein zentraler Begriff der der „Cartridges“. Über diese werden benötigte Funktionalitäten für eine Anwendung bereitgestellt. Man benötigt beispielsweise ein spezielles Sprachen-Cartridge für die benutzte Programmiersprache wie Java EE oder PHP. Außerdem gibt es Cartridges für den Zugriff auf Datenbanken, Services oder auch Frameworks. Zum Ausführen von Software ist ein Cartridge für ein Framework zwingend notwendig, denn dadurch wird festgelegt, wie die Anwendung bereitgestellt wird, also z.B. per JBoss. Durch das Auswählen und Hinzufügen von Cartridges kann es durch Abhängigkeiten dazu kommen, dass weitere Cartridges hinzugefügt werden müssen, denn eine serverseitige Skriptsprache benötigt z.B. ebenfalls einen Webserver. Zur Administration bietet OpenShift den Zugriff durch Clients per REST-API oder auch SSH-Dienst an. Als Entwicklungsumgebung wird das JBoss Developer Studio empfohlen, da dieses an die Plattform von OpenShift gut angebunden werden kann. Das Deployment und die Versionierung kann vom Entwickler per Git realisiert werden.

Der Vorteil von OpenShift für die Projektgruppe ist die kostenlose Möglichkeit, in Java entwickeln und deployen zu können. Ein Nachteil sind fehlende Eingriffsmöglichkeiten bei der Auswahl der Versionen ausführender WebServer im Hinblick auf Sicherheit.

5.1.10. Oracle Cloud

Oracle Cloud ist eine kombinierte Lösung aus den Bereichen SaaS und PaaS [76]. SaaS richtet sich in diesem Fall speziell an Unternehmen, da vornehmlich Funktionen für betriebswirtschaftliche Zwecke zur Verfügung stehen. Auf der anderen Seite ermöglichen der „Oracle Java Cloud Service“ und der „Oracle Database Cloud Service“ als Module das Deployment von Applikationen in der Oracle Cloud als Plattform. Dabei gibt es die volle Unterstützung für Java Enterprise-Applikationen. Eine Besonderheit hierbei ist, dass man keine spezielle APIs nutzen muss wie bei anderen PaaS-Anbietern. Standardtechnologien aus der Java-Welt wie JPA reichen für den Zugriff auf angebotene Dienste aus. Die Entwicklung kann in einer beliebigen IDE vorgenommen werden. Für das Deployment gibt es diverse Plugins.

Bei Oracle Cloud ist es für die Projektgruppe von Vorteil, dass mit Java entwickelt werden kann. Nachteile sind die Tatsachen, dass keine relationalen Datenbanken genutzt werden können. Zu Beginn der Projektgruppe konnten lediglich Ressourcen bei Oracle angefordert werden, allerdings wurden sie nach mehreren Wochen noch immer nicht bereit gestellt. Aus diesem Grund ließ sich der Dienst im Rahmen des Projektes nicht nutzen.

5.2. Diskussion PaaS

Nach der Seminarphase war ein erster Überblick über Möglichkeiten und Rahmenbedingungen einiger Plattform-Lösungen gegeben. Auf dieser Grundlage sollte im Anschluss ein Erproben der Angebote in Hinblick auf praktische Umsetzungen im Rahmen der Projektgruppe stattfinden. Zu diesem Zweck wurde in einem ersten Schritt für jede Plattform eine „Hello, World!“-Applikation erstellt und beim jeweiligen Anbieter ausgeführt. Allerdings wurde die Betrachtung einiger in der Seminarphase behandelter Angebote eingestellt. Die Projektgruppe hat sich nicht weiter mit der Apache Deltacloud beschäftigt. Das liegt darin begründet, dass es sich bei dieser Lösung nicht um PaaS handelt, sondern ein Produkt zum Nutzen und Managen von IaaS-Lösungen. Aus diesem Grund wurde im Folgenden die Apache Deltacloud nicht weiter betrachtet.

Außerdem wurde das Projekt der Apache Libcloud im weiteren Verlauf nicht weiter berücksichtigt. Die Libcloud ist eine Python-Bibliothek, aber die Projektgruppe legte den Fokus auf die Programmiersprache Java. Außerdem ist die Libcloud kein PaaS-Anbieter.

Force.com erwies sich ebenfalls als nicht geeignet für den praktischen Einsatz im Rahmen der Projektgruppe. Für ein größeres Projekt, bei dem Software durch die Teilnehmer selbst erstellt wird, bietet Force.com nicht die nötige Freiheit, da sie die Java Dateien nicht hosten und nur per SOAP darauf zugreifen. Alle Bestandteile, welche in Java realisiert sind, müssten an anderer Stelle gehosted werden. Ein weiterer Kritikpunkt ist die fehlende Skalierbarkeit, eine wichtige Eigenschaft von PaaS.

Da es sich bei den Google Apps Application APIs nicht um eine Lösung zu PaaS handelt, wurden sie im Folgenden nicht weiter betrachtet.

Die Oracle Cloud wurde ebenfalls nicht mehr berücksichtigt. Zum Zeitpunkt der Seminarphase handelte es sich hier um ein Angebot in einer Art Beta-Phase. Es sollte lediglich einen 30-tägigen kostenlosen Test-Account geben, mit dem man den Dienst testen können sollte. Danach wäre laut Informationen des Anbieters nur noch die kostenpflichtige Nutzung möglich gewesen. Aber selbst die kostenlose Nutzung zur Probe konnte nicht realisiert werden. Nach dem Beantragen des Accounts wurde lediglich zugesichert, dass die angefragten Ressourcen im Zeitraum von einigen Monaten zur Verfügung stehen würden. Bis zum Abschluss der Seminarphase waren keine Möglichkeiten gegeben, den Dienst zu testen. Aus diesem Grund wurde die Oracle Cloud nicht weiter betrachtet.

Die restlichen im vorigen Abschnitt genannten Anbieter wurden für den ersten praktischen Test weiter genutzt. Durch die weggefallenen Anbieter wurde ein Ausgleich durch das zusätzliche Betrachten von Windows Azure und HP Cloud geschaffen.

Das Angebot von HP wurde nach dem Versuch, eine „Hello, World!“-Applikation auszuführen, wieder aus dem Fokus der Projektgruppe genommen. Grund dafür war die Tatsache, dass Kredikartendaten angeben musste, um die HP Cloud nutzen zu können. Aus rechtlichen Gründen entschieden sich die Teilnehmer dagegen, unter

diesen Bedingungen mit einem Anbieter zu arbeiten.

Auch Azure wurde nach der Entwicklung der Beispielapplikation nicht weiter betrachtet. Zwar bot Microsoft mit seinem Dienst sehr gute und zuverlässige PaaS Lösungen mit einem Tomcat Webcontainer, Datenbanken und Datenspeicher. Aber der Testzugang ist auf 30 Tage beschränkt und benötigt in jedem Fall Kreditkarteninformationen zum Zugang. Wie bei anderen Anbietern mit diesen Vorraussetzungen wurde Windows Azure somit ausgeschlossen.

Alle verbliebenen Lösungen, also Cloudbees, Google App Engine, Heroku und OpenShift, hatten sich in den Augen der Teilnehmer dafür qualifiziert, weiter betrachtet und später eventuell auch genutzt zu werden. Grund dafür waren die Unterstützung der Anbieter für Java-Anwendungen. Das Deployment ist hier über WAR-Dateien oder mit Hilfe von Git möglich. Durch das Bereitstellen eines Tomcat-Servers schienen diese Produkte gut für den weiteren Projektverlauf geeignet.

Nach erfolgreicher Umsetzung der „Hello, World!“-Applikationen auf den einzelnen Plattformen wurde zur weiteren Überprüfung der Angebote jeweils eine ToDo-Liste erstellt, um mehr Erfahrungen im Umgang mit Ihnen zu bekommen.

5.2.1. Implementierung ToDo-Liste

Gerade zu Beginn der Projektphase ging es in erster Linie darum, die Besonderheiten der einzelnen Anbieter in Erfahrung zu bringen. Es zeigte sich gerade in dieser Hinsicht, dass es für die Umsetzung der Ziele der Projektgruppe nicht ausreichte, sich auf die Angaben der Anbieter zu verlassen. Aus diesen konnten lediglich gewisse Basisinformationen gewonnen werden. Auf die Fragestellung, ob gewisse Dienste auch praktisch genutzt werden können, waren allerdings konkrete Implementierungstests unerlässlich. Erst aufgrund der Erfahrungen, die bei der konkreten Umsetzung gewonnen werden konnten, war es möglich, abschließend einzuschätzen, welches Angebot für die Projektgruppenziele geeignet war.

Ein Beispiel für eine konkrete Aufgabe war die Implementierung einer ToDo-Liste auf einer Webseite. Jedes Gruppenmitglied musste sich damit beschäftigen, wie eine ToDo-Liste in einem bestimmten Cloud-Service implementiert werden konnte. Dadurch konnten Erkenntnisse bezüglich des Bedienkomforts und eventuellen Einschränkungen gewonnen werden. Es zeigte sich, dass das Deployen auf Tomcat-Servern keine großen Schwierigkeiten bereitete. Als problematischer wurden allerdings die DB-Einschränkungen gesehen. Die Datenbank von CloudBees ist auf 5 MB beschränkt. Da allerdings alle Anbieter die Anbindung an externe Datenbanken unterstützen, konnten diese Anforderungen getrennt betrachtet werden. Als Kandidaten für die Bereitstellung von Datenbanken mit ausreichend großem Speicherplatz und akzeptablen Zugriffszeiten kamen Amazon WebServices und Windows Azure in Frage.

5.3. Zusammenfassung PaaS

Die Auseinandersetzung mit PaaS-Angeboten war in erster Linie durch das Ziel der passenden Selektion gekennzeichnet. Es handelte sich dabei um zwei parallele Prozesse, bei denen zum einen die unterschiedlichen Dienste untersucht wurden und zum anderen der Frage nachgegangen wurde, welche Anforderungen überhaupt gestellt werden müssen. Die zentrale Frage war, wie die zu realisierende Applikation aussehen sollte und welche Programmiersprache sie einsetzt. Da es im Vorfeld keine Einschränkungen gab, wären auch Sprachen wie Python in Frage gekommen.

Die Untersuchung der einzelnen Anbieter zeigte jedoch, dass bei den vielfältigen und zahlreichen PaaS-Angeboten häufig ein Webserver zur Verfügung steht, der das Betreiben einer Webanwendung ermöglicht. Der Umfang der Zugriffs- und Auswahlmöglichkeiten variierte jedoch stark. So bot z.B. die Google App Engine nur eine Art fest vorgegebenes „Framework“ für die Implentierung von Web-Services an. Solche Angebote wurden unabhängig von ihrer Fortschrittlichkeit und Verbreitung im Entscheidungsprozess aussortiert.

Es blieben die Anbieter übrig, die Lösungen anbieten, die nicht das sogenannte „Lock-In“-Problem mit sich bringen. Dabei handelt es sich um Angebote, die statt auf proprietäre Lösungen auf weit verbreitete Standard-Techniken setzen. Mit der Idee eine HTML-Webanwendung zu entwerfen, kristallisierte sich heraus, dass dafür nur die Verwendung eines Web-Servers in Frage kommen konnte, in dem WAR-Archive deployt werden können. Einen solchen Webcontainer stellen die meisten Cloud-Anbieter auch bereit.

Eine übergeordnete Anforderung war die Möglichkeit der kostenlosen Nutzung, die nicht bei allen Anbietern gegeben war. Somit erfolgte die Auswahl nicht allein aufgrund technischer Mermale.

6. Cloud-Storage und Storage-API

In diesem Kapitel wird die Entwicklung der sogenannten „Storage-API“ dargestellt. Dies war einer der Schwerpunkte der Projektgruppe, wie im weiteren Verlauf näher erläutert wird. Es handelt sich hierbei um eine Java-Bibliothek, welche sehr einfach in Entwicklungsprojekte eingebunden werden kann. Sie stellt eine Schnittstelle für die Dateiverwaltung auf unterschiedlichen Cloud-Storage-Plattformen dar.

Die schrittweise Erarbeitung dieses Themas durch die Gruppe spiegelt sich in der Struktur dieses Kapitels wider. Zunächst diskutierte die Gruppe, inwiefern die Entwicklung der Storage-API dem Ziel der PG entspricht. Anschließend wurden unterschiedliche Anbieter verglichen und die drei geeignetsten für die weitere Entwicklung ausgewählt. Dazu mussten zuvor Anforderungen, d.h. von der API unterstützte Dateioperationen sowie minimale Anforderungen an die Performanz der Angebote, aufgestellt werden. Anschließend begann die eigentliche Programmierung der Schnittstelle. Gemäß den Zielen und Aufgaben der API musste im ersten Schritt definiert werden, wie die API im fertigen Zustand zu verwenden sein solle. Daraus entwickelte sich die Code-Struktur und die Technologien, die bei der Implementierung verwendet wurden. Die Einzelheiten der Programmierung und der dabei aufgetretenen Schwierigkeiten sind im Folgenden beschrieben. Nachdem eine erste Version der Schnittstelle vorlag, wurde sie ausgiebig getestet, wodurch einige Schwachstellen entdeckt und behoben wurden. Über das Testen wird im letzten Abschnitt dieses Kapitels berichtet.

6.1. Motivation

Neben der Herausforderung, eine Webapplikation bei verschiedenen PaaS-Anbietern zu deployen, war die Verteilung von Dateien auf verschiedene Cloud-Storage-Plattformen ein weiterer Aspekt, der im Zusammenhang mit dem Thema Intercloud untersucht wurde. Dies entstand aus der Frage, wie segmentiert eine solche Anwendung auf verschiedenen Plattformen verteilt werden kann. So kann z.B. die eigentliche Anwendung auf dem Webserver eines Anbieters ausgeführt werden, während die Datenbank(en) und Dateien, die verwaltet werden, woanders gelagert sind.

Bei der Diskussion über die Arbeit und Schwerpunkte der Projektgruppe wurde entschieden, dass eine Möglichkeit geschaffen werden solle, in einer Webanwendung auszuwählen, bei welchem Storage-Cloud-Anbieter Dateien hinterlegt werden sollen.

In den Treffen herrschte Einigkeit darüber, dass neben der Möglichkeit, eine Webapplikation bei verschiedenen Plattform-Anbietern ausführen zu können, eine einheitliche, flexible Schnittstelle für unterschiedliche Storage-Angebote ein wesentlicher Teil des Zieles der Projektgruppe sei.

Insbesondere eine Webapplikation wie Content-Battle erfordert mehr Storage, je mehr sie genutzt wird. User können hier Content hochladen und gegen den Content anderer User antreten lassen (vgl. Kapitel 7). Es ist daher von Bedeutung, dass der Anwendung ausreichend und gegebenenfalls flexibler Speicherplatz für die von den Usern hochgeladenen Dateien zur Verfügung steht. Tatsächlich war dies ein Aspekt,

der bei der Entwicklung des Konzeptes für das Testprojekt eine wesentliche Rolle einnahm.

Die Gruppe entschied sich, eine einheitliche Schnittstelle zu entwickeln, über die Dateioperationen auf verschiedenen Storage-Plattformen durchgeführt werden können. Da die Unterstützung aller bekannten und unbekannten Anbieter weit über das Ziel der Projektgruppe hinausgegangen wäre, sollten drei geeignete Plattformen ausgewählt werden. Zusätzlich wurde bestimmt, dass zu Testzwecken ebenfalls eine lokale Dateiverwaltung implementiert werden sollte. Diese kann zudem in einer Intercloud-Applikation als Zwischenspeicher genutzt werden, um etwaige unzumutbare Wartezeiten zu vermeiden.

Am Ende sollte für einige ausgewählte Anbieter eine einheitliche Storage-API entwickelt werden, die als Bibliothek in das Beispielprojekt eingebunden werden kann.

6.2. Auswahl der Anbieter

Im Folgenden wird beschrieben, welche einzelnen Storage-Anbieter von der Gruppe untersucht und nach welchen Kriterien sie anschließend für das weitere Vorgehen ausgewählt wurden.

6.2.1. Kriterien

Die Auswahl der geeigneten Storage-Anbieter gliederte sich, wie schon bei der Wahl der PaaS-Lösungen, in zwei Phasen. Zunächst wurden verschiedene Anbieter von den Gruppenmitgliedern untersucht und auf ihre Eignung hin geprüft. Dafür wurde zuvor eine Liste notwendiger Funktionen und Operationen aufgestellt, die jeder Anbieter unterstützen sollte.

Dies waren:

- Login über Schnittstelle
- Datei-Download, Datei-Upload
- Dateien löschen, verschieben, umbenennen
- Ordner erstellen, löschen, verschieben, umbenennen
- Metadaten (Name, Mime-Type etc.) von Dateien und Ordnern abrufen

Neben den Dateioperationen war zudem die Frage entscheidend, ob die Anbieter Direct-Downloads, d.h. direkte Links auf Dateien statt auf eine spezielle Download-Seite, unterstützen. Solche Download-Seiten benutzt z.B. der Anbieter „Dropbox“. Zwar bestand die Möglichkeit, dieses Problem durch Workarounds zu umgehen, jedoch war unklar, ob dies rechtlich gesehen unbedenklich ist, da die entsprechenden Anbieter diese Download-Seiten erzwingen, um Werbung einzublenden. Es ist somit nicht im Sinne der Anbieter, diese Seiten zu umgehen. Aus diesem Grund sahen wir

schließlich davon ab und wählten für die Entwicklung der Storage-API lediglich solche Anbieter, die Direct-Links oder Filestreams unterstützen. Das Gleiche galt für die Authentifizierung. Einige Anbieter erlaubten den Zugriff nur dann, wenn eine manuelle Anmeldung über ein Webformular erfolgte. Eine entsprechende Schnittstelle wie OAuth stellten diese Anbieter dagegen nicht bereit. Auch dies ließe sich umgehen, indem etwa ein Workaround benutzt wird, das z.B. Browseraktionen automatisiert. Jedoch vertrat die Gruppe die einhellige Meinung, dass eine solche Lösung nicht ideal ist und lediglich gewählt würde, wenn es keine andere Möglichkeit gäbe.

Darüber hinaus spielten auch die Preise eine entscheidende Rolle bei der Auswahl der Anbieter. Unter der Annahme, dass ausreichend kostenlose Anbieter existieren, wurden von vornherein ausschließlich diese untersucht. Hätte sich im Laufe der Untersuchungen herausgestellt, dass nicht ausreichend kostenlose Angebote geeignet sind, hätten wir sicherlich über kommerzielle Lösungen nachdenken müssen.

6.2.2. Anbieter

Nachdem die Kriterien festgelegt waren, wurden mehrere Storage-Anbieter ausgesucht. Jedes Gruppenmitglied übernahm anschließend einen Anbieter zwecks näherer Untersuchung. Dabei wurde das Beispielprojekt, welches bereits bei den Tests der PaaS-Anbieter genutzt wurde - eine einfache, selbstgeschriebene Datenbankanwendung in Form einer ToDo-Liste (siehe Abschnitt 5.2.1) - um den Zugriff auf Dateien beim jeweiligen Storage-Anbieter erweitert. Im Folgenden werden diese Anbieter kurz dargestellt.

- **Box** - www.box.com
- **Dropbox** - www.dropbox.com
- **Microsoft SkyDrive** - www.windowslive.de/skydrive/
- **Google Drive** - drive.google.com
- **Mediafire** - www.mediafire.com
- **SugarSync** - www.sugarsync.com
- **4shared** - www.4shared.com
- **SpiderOak** - www.spideroak.com

Box Box bietet 5 GB Speicher kostenlos an. Außerdem steht eine Java API inklusive eines Tutorials zur Verwendung bereit [38]. Des Weiteren gibt es eine sehr gute Dokumentation auf der Produktseite [40]. Über die API werden alle gewünschten Datei- und Ordneroperationen ermöglicht. Bei den Tests mit der ToDo-Liste erwies sich die Geschwindigkeit des Zugriffs als ausreichend. Jedoch stellte sich heraus, dass die Authentifizierung ausschließlich über die manuelle Eingabe der Benutzerdaten in ein Webformular erfolgen kann. Die Möglichkeit einer Anmeldung über die API besteht nicht.

Dropbox Dropbox zählt mit über 175 Millionen Nutzern [54] zu den verbreitetsten Anbietern für Cloud-Storage, bietet jedoch nur 2 GB kostenfrei an. Entwicklern stellt Dropbox APIs für viele verschiedene Programmier- und Skriptsprachen zur Verfügung [10]. Die Kommunikation erfolgt dabei über eine REST-Schnittstelle. Ein Java-SDK - ursprünglich für Android-Entwickler gedacht - wird ebenfalls angeboten. Beim Test zeigte sich allerdings, dass die Geschwindigkeit der Dateioperationen weit hinter der anderer Anbieter zurück blieb. Noch gravierender war das Problem, dass keine Direct-Links erzeugt werden konnten. Für den Download einer Datei wurde lediglich ein Link auf eine gesonderte Download-Seite ausgegeben. Wie bereits zuvor geschildert, wurde über ein Workaround diskutiert, jedoch erwies sich der Aufwand sowie rechtliche Bedenken als zu groß.

Microsoft SkyDrive SkyDrive heißt der Cloud-Dienst von Microsoft. Er bot anfangs 25 GB Speicherplatz. Inzwischen sind es nur noch 7 GB, die gratis zur Verfügung stehen. SkyDrive stellt eine JavaScript API sowie eine für die Windows Live Dienste bereit [32]. Der Grundgedanke hinter der Microsoft Cloud zielt in einer andere Richtung ab, als es die Projektgruppe vorsieht. SkyDrive ist als private Cloud angelegt, um z.B. Dateien sowohl zu Hause als auch am Arbeitsplatz zur Verfügung zu haben oder unterschiedliche Windows (bzw. Live-fähige) Geräte untereinander zu synchronisieren. Der öffentliche Zugang zu den Daten ist dabei nicht vorgesehen. Für diese Zwecke wurde der kostenpflichtige Dienst Windows Azure gestartet.

Google Drive Google Drive löste im letzten Jahr den Dienst Google Docs ab und verfügt in der kostenlosen Variante über 5 GB Speicherkapazität. Es werden Client Libraries für verschiedene Sprachen angeboten, darunter auch Java [13]. Sie sind gut dokumentiert und erlauben alle von uns gewünschten Operationen. Die Authentifizierung erfolgt über OAuth. Es ist jedoch erforderlich, die Anwendung im Vorfeld bei Google zu registrieren.

Mediafire Mit 50 GB kostenlosem Speicher bietet Mediafire die größte Kapazität. Auch hier gibt es eine REST-Schnittstelle. Die Dokumentation ist stellenweise unvollständig, jedoch werden alle benötigten Funktionen von der Schnittstelle unterstützt. Für die Authentifizierung kommt OAuth zum Einsatz. Schon bei den ersten Tests zeigte sich, dass die Verbindungsgeschwindigkeit äußerst gering ist. Zudem ist das Downloadvolumen begrenzt, nach Ansicht der Gruppe stellte dies für das Testprojekt aber keine gravierende Einschränkung dar. Diese Einschätzung erwies sich im Nachhinein als richtig.

SugarSync Beim Anbieter SugarSync bekommt man 5 GB kostenlos. Mittels einer angebotenen REST-basierten API sind grundlegende Dateioperationen möglich. Beim Test stellte sich heraus, dass Teile der API fehlerhaft sind. Außerdem ist die Dokumentation unübersichtlich und stellte bei den Problemen keine Hilfe dar.

4shared Ein kostenloser Account bei 4shared verfügt über 15 GB Speicherkapazität. Für Entwickler bietet 4shared eine REST-Schnittstelle, die allerdings in den Tests in Verbindung mit OAuth nicht funktionierte (siehe Abschnitt 6.4.5). Darüber hinaus gibt es jedoch auch eine SOAP-Schnittstelle. Diese kann in Form einer JAR-Datei heruntergeladen und in Java-Projekte eingebunden werden. Um die originalen Klassen benutzbar zu machen, mussten einige Fehler nachgebessert werden. Die API erfüllt alle im Vorfeld festgelegten Bedingungen und ist ausreichend performant. Auch bei 4shared gibt es das Problem, dass Direct-Downloads nicht möglich sind, allerdings können Dateien über die API als Streams direkt heruntergeladen werden.

SpiderOak SpiderOak ist ein Storage-Dienst, der sich auf die Sicherung von Daten spezialisiert hat. Der Fokus liegt auf der Sicherheit durch konsequente Verschlüsselung aller übermittelten Dateien anstatt auf Kollaboration. Damit eignet sich SpiderOak als Online-Tresor für kritische Daten. Ein kostenloser Account verfügt über 2 Gigabyte Speicher. In den Tests erwies sich das Angebot schon über die Weboberfläche als außerordentlich langsam. Ähnliche Erfahrungen machten wir mit der sog. SpiderOak Web API [33]. Außerdem fehlen hier einige der benötigten Dateioperationen wie z.B. „verschieben“ oder „Metadaten abrufen“, die aufgrund des Sicherheitskonzeptes auf den verschlüsselten Dateien offenbar auch nicht gewollt sind. Außerdem müssen die Schlüssel zusätzlich verwaltet werden. Insgesamt ist dieses Angebot daher für unsere Zwecke nicht geeignet.

6.2.3. Weitere Alternativen

Neben den auf Storage-Lösungen spezialisierten Anbietern wurden im Rahmen der Projektgruppe auch noch andere Möglichkeiten für die Storage-API in Betracht gezogen.

Lokaler Storage Die Möglichkeit wurde zuvor bereits erörtert. Die Dateien werden hier unmittelbar auf dem Application-Server gespeichert, auf dem die Webanwendung ausgeführt wird. Diese Lösung ist besonders performant und kann daher auch zum Zwischenspeichern von Dateien verwendet werden.

OpenStack Zwischenzeitlich wurde auch eine Lösung mit OpenStack [25] diskutiert. Tatsächlich bietet ein OpenStack-Server alle geforderten Methoden, abgesehen von der Move-Operation. Maik und Markus loteten die Möglichkeit aus, eine solche OpenStack-Lösung am Lehrstuhl zu installieren. Allerdings wäre dies mit zu vielen technischen Komplikationen verbunden gewesen, weshalb von einem Storage auf Basis von OpenStack wieder abgesehen wurde.

6.2.4. Diskussion

Aufbauend auf den im vorherigen Abschnitt beschriebenen Erkenntnissen musste eine Wahl getroffen werden, welche Anbieter für die Intercloud-Storage-API infrage kamen und welche nicht. Nach Meinung der Betreuer Maik und Markus wären drei oder vier Anbieter im Rahmen der Projektgruppe vollkommen ausreichend. Die Gruppe entschied daher, sich auf diese Anzahl festzulegen.

Dabei wurden die zuvor erläuterten Pro- und Contra-Argumente der einzelnen Anbieter gegeneinander abgewogen. Es fiel auf, dass so gut wie keine Lösung optimal war. Bei nahezu jedem Anbieter mussten kleinere oder größere Einschränkungen in Kauf genommen werden. Den besten Eindruck hat Google Drive gemacht. Der Dienst bietet nach kurzen Tests im Rahmen der Evaluation eine gute Performanz, eine übersichtlich ausführliche Dokumentation und vor allem erfüllte die API alle an sie gestellten Erwartungen. Aufgrund der Popularität des Dienstes und der Marke Google, fiel die Entscheidung für Google Drive sehr schnell.

Von den anderen Anbietern wurden 4shared und Mediafire ausgewählt. 4shared wurde wegen seiner SOAP-Schnittstelle angenommen und Mediafire aufgrund des sehr großen Speicherplatzes und der Tatsache, dass die API alle geforderten Operationen im Wesentlichen unterstützte. Diese Gründe überwogen den Fakt, dass die Performanz in unseren Tests bei Mediafire nur gering war.

Zu guter Letzt wurde noch eine lokale Implementierung beschlossen.

Die anderen Angebote wiesen dagegen Faktoren auf, die die Gruppe zur Entscheidung gegen diese bewogen. Entweder war es die fehlende Möglichkeit eines Direct-Downloads (bei Dropbox) oder aber die Tatsache, dass die Authentifizierung nur manuell durch den User über ein Webformular erfolgen konnte (bei Box). Die Möglichkeit, entsprechende Workarounds zu schreiben, wurde zwar diskutiert, aber aufgrund der zusätzlichen Arbeit und rechtlicher Bedenken verworfen. Die API von SugarSync wirkte sehr unausgereift und war fehlerhaft, zudem war sie auch nur unzureichend dokumentiert. Dies disqualifizierte SugarSync für uns. Gegenüber den konventionellen Storage-Anbietern verfolgte SpiderOak einen komplett anderen Ansatz und sah sich mehr als Datentresor. Dies führt zu Schwierigkeiten bei der Schlüsselverwaltung und Performanz. Daher schied SpiderOak von vornherein aus. Auch Microsofts SkyDrive wurde ausgeschlossen, aber aus dem Grund, das sich dieser Dienst eher als private Cloud statt als allgemein zugänglicher Speicherdienst versteht.

6.3. Planung

Die erste Frage, die für die Storage-Schnittstelle geklärt werden musste, war, welche Funktionen die API anbieten sollte. Bei der Klärung dieser Frage wurde bereits berücksichtigt, was im späteren Verlauf der Projektgruppe für ein internes Testprojekt benötigt werden könnte. Durch die Gruppe wurde eine Liste an Funktionen zusammengetragen, welche die Storage-Schnittstelle unterstützen sollte. Dabei orientierte man sich stark an bestehenden, bekannten Funktionen eines Dateisystems.

Außerdem wurden Gedanken diskutiert, wie man sich gegenüber dem Anbieter authentifiziert.

Die folgende Auflistung zeigt den ersten Entwurf der geplanten Funktionen:

- Authentifizierung
- Dateien
  - Hochladen, Runterladen
  - Umbenennen, Verschieben, Löschen
  - Metadaten abrufen
- Ordner
  - Erstellen, Löschen, Verschieben, Umbenennen
  - Metadaten abrufen
- Metadaten abrufen

Neben den Überlegungen zu den geplanten Funktionen der Storage-Schnittstelle wurde darüber diskutiert, welche Webservicetechnologie zur Anbindung der Storage-Schnittstelle verwendet werden soll. Zur Diskussion standen REST und SOAP. Geeinigt wurde sich auf die Verwendung der REST-Schnittstelle der einzelnen Anbieter, obwohl das bei 4Shared nur mit Hilfe eines kleinen Umwegs möglich ist. Alle anderen Anbieter boten hingegen bereits eine REST-Schnittstelle an, wodurch die Umsetzung vereinfacht wurde. Im späteren Verlauf der Entwicklung der Storage-Schnittstelle wurde bei 4Shared die Umsetzung über SOAP anstatt REST bevorzugt. Mehr dazu ist in Kapitel 6.4.5 nachlesbar.

Auf den ersten Entwurf folgte eine weitere Überprüfung der Anbieter, ob und wie sich diese Funktionen umsetzen lassen. Dabei stellte sich heraus, dass für Google Drive und 4Shared alle geplanten Funktionen umsetzbar sind. Allerdings stellte sich auch heraus, dass bei MediaFire die Abfrage von Metadaten nicht direkt möglich ist. Dies wurde jedoch nicht als Problem angesehen. Für eine lokale Umsetzung lagen ebenfalls keine Probleme vor. Des Weiteren erwies sich eine konkrete Methode zur Authentifizierung als nutzlos, da eine Authentifizierung nur intern erfolgen muss (zum Beispiel beim Instanziieren der Schnittstelle).

Der finale Entwurf der Funktionen entspricht damit dem ersten Entwurf, jedoch ohne direkte Funktion zur Authentifizierung.

6.4. Implementierung

Nachdem der gewünschte Funktionsumfang der Storage-Schnittstelle festgelegt wurde, musste das Ziel der Entwicklung dieser Schnittstelle definiert werden. Da sie später Verwendung in mehreren Projekten finden sollte, erfolgte in Bezug auf diese Frage die Einigung auf ein JAR-Archiv. Um eine einfache Verwendung und Einbindung zu gewährleisten und um die Implementierung zu beschleunigen, einigte man

sich außerdem darauf, Maven zu verwenden. Über die `pom.xml` wurden die folgenden Eigenschaften für die Storage-Schnittstelle festgelegt:

- Name (intercloud-storage) und Version der Storage-Schnittstelle
- Verwendet wird Java 1.6
- Ausgabe des Compiliervorgangs ist ein JAR-Archiv
- Der Compiliervorgang exportiert sowohl den Sourcecode als auch das Javadoc
- Die einzelnen verwendeten Bibliotheken werden definiert

Um die Schnittstelle später mittels Maven importieren zu können, muss sie in einem Repository angeboten werden. Dazu wurde auf `github.com` ein Maven Repository aufgesetzt und ebenfalls in der Konfigurationsdatei eingestellt. Die Urls des eingerichteten Repositories sind `https://github.com/herrdommel/intercloud-maven-repo/raw/master/releases` und `https://github.com/herrdommel/intercloud-maven-repo/raw/master/snapshots` für das Release- und das Snapshot-Repository.

6.4.1. Interface

Um eine bessere Benutzbarkeit der Storage-Schnittstelle zu gewährleisten, wurde festgelegt, dass zunächst Interfaces definiert wurden, welche dann von den einzelnen Storage-Komponenten implementiert werden mussten. Dabei wurden drei Interfaces definiert.

IStorageInfo Das Interface `IStorageInfo` beinhaltet die relevanten Methoden, um die Metadaten des Storageanbieters abzufragen.

Die Methode `String getSpaceQuota()` ermöglicht das Abrufen des für den Besitzer verfügbaren Speicherplatzes, während die Methode `String getUsedSpace()` den bereits belegten Speicherplatz zurückgibt. Die dritte Methode im IStorageInfo Interface ist `String getDownloadQuota()`. Sie ist speziell für Mediafire angedacht und gibt die noch verfügbare Downloadquota zurück. Für Anbieter, die die Downloadquota nicht begrenzen, wird -1 zurückgegeben.

IMetaData Dieses Interface ist für die Metadaten einzelner Ordner und Dateien angedacht. Es beinhaltet insgesamt neun Methoden, um einzelne Metadaten abzufragen und zu setzen. Dazu sind jeweils vier Getter- und vier Settermethoden zur Manipulation vorhanden.

- Mime Type vom Typ String
  - `String getMimeType();`
  - `void setMimeType(String mimeType);`

- Dateiname vom Typ String
  - `String getFileName();`
  - `void setFileName(String fileName);`
- Id vom Typ String
  - `String getId();`
  - `void setId(String id);`
- Dateigröße vom Typ Long
  - `Long getFileSize();`
  - `void setFileSize(Long fileSize);`

Außerdem beinhaltet das Interface `IMetaData` die Methode `isDirectory()`, um zu überprüfen, ob die Metadaten zu einem Ordner oder zu einer Datei gehören. Im ersten Fall liefert die Methode `true` und im zweiten Fall `false` zurück.

IStorage Dieses Interface bildet die Hauptkomponente der jeweiligen Implementierung. Es enthält Methoden, um die in Kapitel 6.3 definierten Funktionen umsetzen zu können.

In Listing 1 sind die Methoden für Operationen auf Dateien dargestellt. Die Methoden `uploadFile` und `downloadFile` arbeiten mit Hilfe von `InputStreams`, da einige PaaS Anbieter das Erstellen von Dateien auf ihren eigenen Servern nicht zulassen, jedoch alle Storageanbieter mit `InputStreams` umgehen können. Während die Methode `uploadFile` die Id der hochgeladenen Datei in Form eines `Strings` zurückgibt, lässt sich mit dieser Id und der Methode `downloadFile` diese Datei wieder herunterladen. Die drei weiteren Methoden für Dateioperationen verwenden ebenfalls diese Id vom Typ `String`. Des Weiteren benötigt die Methode `moveFile` einen Zielordner, in welchen die Datei verschoben werden soll. Die Methode `modifyFile` verarbeitet ein Metadatenobjekt das die Änderungen wiederspiegelt.

```
String uploadFile(InputStream in, IMetaData meta,
    String parentId) throws StorageProviderException;
InputStream downloadFile(String fileId)
    throws StorageProviderException;
void moveFile(String fileId, String parentDestination)
    throws StorageProviderException;
void deleteFile(String fileId)
    throws StorageProviderException;
void modifyFile(String fileId, IMetaData meta)
    throws StorageProviderException;
```

Listing 1: Methodendefinition für Dateioperationen

Die Methoden für Operationen auf Ordnern sind in Listing 2 dargestellt. Genau wie die Methoden für Dateioperationen verwenden sie Ids, um die Ordner zu identifizieren. Die Methoden `createFolder` und `updateFolder` benötigen ein Metadatenobjekt, welches die Eigenschaften des Ordners, wie zum Beispiel den Namen, spezifiziert. Die Methode `listFolder` listet alle in dem mit `folderId` spezifizierten Ordner vorhandenen Dateien und Unterordner. Dabei enthält die zurückgegebene Liste für jeden Unterordner und jede Datei ein Metadatenobjekt.

```
String createFolder(IMetaData meta, String parentId)
    throws StorageProviderException;
void deleteFolder(String folderId)
    throws StorageProviderException;
List<IMetaData> listFolder(String folderId)
    throws StorageProviderException;
void updateFolder(String folderId, IMetaData meta)
    throws StorageProviderException;
```

Listing 2: Methodendefinition für Ordneroperationen

Während die bereits genannten Methoden sich auf Ordner und Dateien unterteilen lassen, gibt es noch drei weitere Methoden, für die das nicht der Fall ist. Dabei handelt es sich um die Methode `getFileHeader`, welche zu einem mittels `fileId` spezifizierten Ordner oder Datei ein Metadatenobjekt zurückliefert und die Methoden `getStorageInfo` und `getStorageProviderName`. `getStorageInfo` liefert allgemeine Informationen zum verwendeten Storageanbieter und `getStorageProviderName` gibt lediglich den Namen des verwendeten Storageanbieters zurück.

```
IMetaData getFileHeader(String fileId)
    throws StorageProviderException;
IStorageInfo getStorageInfo()
    throws StorageProviderException;
String getStorageProviderName();
```

Listing 3: Methodendefinition für allgemeine Operationen

Ausnahmebehandlung Zusätzlich zu den Methoden werden in den einzelnen Interfaces Exceptions definiert, die von den einzelnen Methoden geworfen werden können. Dabei handelt es sich um die `StorageProviderException`. Die `StorageProviderException` stellt eine Oberklasse dar, die von der Klasse `StorageIOException` erweitert wird. Diese Klasse beinhaltet das Enum `StorageIOErrors`, welches mögliche Fehler definiert. Die folgenden Fehlerfälle können damit dokumentiert werden:

FILE_UPLOAD Eine Datei konnte nicht hochgeladen werden.

FILE_DOWNLOAD Es ist ein Fehler beim Herunterladen der Datei aufgetreten.

FILE_INFORMATION Die Metadaten der Datei konnten nicht gelesen werden.

FILE_MOVE Die Datei konnte nicht verschoben werden.

FILE_NOT_FOUND Die Datei wurde nicht gefunden.

FILE_DELETE Die Datei konnte nicht gelöscht werden.

FILE_MODIFY_METADATA Die Metadaten konnten nicht aktualisiert werden.

FILE_DOWNLOADLINK Es ist ein Fehler aufgetreten, während eine Datei mittels Downloadlink abgefragt wurde.

FOLDER_CONTENT Der Inhalt eines Ordners konnte nicht abgerufen werden.

FOLDER_INFORMATION Die Metadaten eines Ordners konnten nicht abgerufen werden.

FOLDER_CREATE Der Ordner konnte nicht erstellt werden.

FOLDER_DELETE Der Ordner konnte nicht gelöscht werden.

FOLDER_MODIFY_METADATA Die Metadaten des Ordners konnten nicht aktualisiert werden.

STORAGE_INFORMATION Das Abfragen der Informationen des Storageanbieters hat einen Fehler verursacht.

Zusätzlich lassen sich zu den definierten Fehlerfällen weitere Informationen in der `StorageIOException` Klasse speichern.

6.4.2. Google Drive Implementierung

Die Google Drive Storage Implementierung basiert auf dem von Google angebotenen Service *Google Drive*. Google bietet aktuell[2] zwei Versionen der Schnittstelle für Google Drive an, die Version v1 und v2. Die Version v1 ist jedoch überholt und es wird empfohlen die Version v2 zu verwenden. Dementsprechend basiert die Storage-Schnittstelle auf dem für Version v2 angebotenen Software Development Kit. Google bietet neben diesem SDK noch weitere SDKs für die meisten seiner Services an, jedoch werden für die Storage-Schnittstelle keine weiteren benötigt. Um mit dem Google Drive Service zu kommunizieren, kann eine REST Schnittstelle programmatisch angesprochen oder stattdessen das Google Drive SDK verwendet werden. Letzteres kapselt die Aufrufe und stellt die benötigten Klassen zur Verfügung. Des Weiteren stellt Google für das SDK eine Dokumentation bereit [14]. Die Dokumentation enthält neben wichtigen Informationen zur Authentifizierung auch Informationen zu den einzelnen Anfragen, die Google Drive unterstützt.

[2]Stand: 20.03.2012

Authentifizierung Für die Authentifizierung stellt Google Drive mehrere Möglichkeiten zur Verfügung. Die von uns eingesetzte Methode besteht in der Verwendung von sogenannten *Service Accounts*. Ein Service Account ist dadurch definiert, dass er keinem Benutzer direkt gehört, sondern zu einem Service beziehungsweise Projekt. Dieser Account lässt sich nur programmatisch verwenden. Dazu benötigt man die dem Account zugewiesene Service-Email-Adresse und einen generierten Private-Key im PKCS#12 Format. Beides erzeugt Google Drive während des Anlegens des Service Accounts. Die Prozedur, die die Google Drive Storage Implementierung verwendet, ist beispielhaft in Listing 4 dargestellt. Dabei sind `SERVICE_ACCOUNT_EMAIL` und `PRIVATE_KEY_FILE_PATH` die bereits angesprochenen benötigten Eingaben. Das in der letzten Zeile erstelle `Drive` Objekt stellt den nun verwendbaren Service dar, um mit dem Google Drive Service zu kommunizieren.

```
HttpTransport httpTransport = new NetHttpTransport();
JacksonFactory jsonFactory = new JacksonFactory();
GoogleCredential credential = new GoogleCredential
    .Builder()
    .setTransport(httpTransport)
    .setJsonFactory(jsonFactory)
    .setServiceAccountId(SERVICE_ACCOUNT_EMAIL)
    .setServiceAccountScopes(DriveScopes.DRIVE)
    .setServiceAccountPrivateKeyFromP12File(
      new java.io.File(PRIVATE_KEY_FILE_PATH))
    .build();
Drive service = new Drive.Builder(httpTransport,
   jsonFactory, null)
   .setHttpRequestInitializer(credential).build();
```

Listing 4: Authentifizierungsprozedur für Google Drive mithilfe eines Service Accounts

Dateitransfer Das Hoch- beziehungsweise Herunterladen von Dateien von Google Drive gestaltet sich mit dem SDK denkbar einfach. Zum Hochladen von Contents wird zunächst ein `com.google.api.services.drive.model.File` Objekt und ein `com.google.api.client.http.InputStreamContent` Objekt erstellt. Diese werden mit dem übergebenen Inhalt gefüllt und über die `insert()` Methode des Drive Services hochgeladen. Die Prozedur ist in Listing 5 dargestellt.

Das Herunterladen von Dateien anhand eines Downloadverweises ist sogar noch einfacher. Dazu wird mittels des SDK ein Get-Request erstellt und ausgeführt. Die Antwort dieser Anfrage enthält die gewünschte Datei. Dies ist beispielhaft in Listing 6 dokumentiert.

```
File body = new File();
body.setTitle(title);
body.setMimeType(mimeType);
InputStreamContent mediaContent =
    new InputStreamContent(mimeType, content);
mediaContent.setLength(fileLength);
Insert insert = service.files()
  .insert(body, mediaContent);
insert.getMediaHttpUploader()
  .setDirectUploadEnabled(true);
File file = insert.execute();
```

Listing 5: Hochladen einer Datei zu Google Drive

```
HttpResponse resp = service.getRequestFactory()
  .buildGetRequest(new GenericUrl(downloadUrl))
    .execute();
InputStream out = resp.getContent();
```

Listing 6: Herunterladen einer Datei von Google Drive

Metadaten Der Google Drive Service bietet eine erweiterte Form der Metadaten an, als im Interface `IMetaData` vorgegeben. Diese könnten später bei einer exklusiven Verwendung von Google Drive benutzt werden. Dabei handelt es sich um eine Download Url, eine alternative Download Url, das Erstellungsdatum sowie Ordner Id.

6.4.3. Lokale Implementierung

Um bei der späteren Entwicklung keine großen Wartezeiten für das Hoch- bzw. Herunterladen von Dateien zu haben, war es sinnvoll, einen fiktiven Storage-Provider zu entwickeln, der zwar die entworfenen Interfaces implementiert, jedoch nicht wirklich Dateien in eine Cloud lädt, sondern diese auf der lokalen Festplatte des Benutzers abspeichert. Es wurde die Klasse `LocalStorage` implementiert, in dessen Konstruktor angegeben werden kann, in welchem Ordner der Cloud-Storage simuliert werden soll. Der Kern hinter dieser Implementierung ist eine sogenannte `DualHashBidiMap`, welche die Funktionalitäten einer normalen `HashMap` darum erweitert, dass es neben der Möglichkeit den Value über den Key zu finden, auch möglich ist, umgekehrt den Key über den Value zu finden. In dieser Map werden als Key UUIDs gespeichert, deren zugehöriger Value ein Pfad auf eine Datei oder einen Ordner ist. Wird eine Datei per `uploadFile` in den LocalStorage geladen, bekommt der Nutzer eine UUID zurück, welche dann in der `DualHashBidiMap` gespeichert wird. Auf diese Weise ist etwa für ein späteres Downloaden stets die Verknüpfung der Datei zu ihrer ID gegeben. Eine aktuelle Version der Map wird stets serialisiert und auf die

Festplatte gespeichert, um so auch bei einem Neustart des Programms die bereits angelegte „Cloud“ und deren IDs nicht zu verlieren. Die Möglichkeit, auf Inhalte in der Map anhand des Wertes zuzugreifen, ist erforderlich, weil bei einem rekursiven Löschen von Ordnern nicht die Keys der Unterordner/Dateien bekannt sind, sondern nur die Pfade, welche die Values darstellen. Die Speicherung der Metadaten erfolgt in serparaten Dateien, die für jede hochgeladene Datei zusätzlich angelegt werden. Diese sind benannt wie die eigentliche Datei, nur haben sie die zusätzliche Endung „.metadata“ (z.B. test.txt und test.txt.metadata).

6.4.4. Mediafire Implementierung

Mediafire stellt für Entwickler nur eine unzureichend dokumentierte REST-Schnittstelle zur Verfügung, um den Zugriff auf Daten direkt aus einer Applikation heraus zu steuern. Die Dokumentation beinhaltet eine kurze Schnittstellenbeschreibung, bestehend aus den zu übermittelnden Parametern und dem Aufbau der Telegramme. Bei der Liste der Parameter hatten sich leider einige Fehler eingeschlichen, was die Anbindung verkomplizierte. Mit 50 GB kostenlosem Speicherplatz bekommt man zwar mehr als ausreichend Platz zur Verfügung gestellt, allerdings ist die Übertragungsgeschwindigkeit zeitweise sehr gering. Dies machte sich bereits bei den Tests negativ bemerkbar. Da die Schnittstelle aber voll funktionstüchtig ist, wurde der Anbieter im Projekt belassen. Bei der Implementierung konnten die meisten im Interface festgelegten Methoden direkt umgesetzt werden. Wo Probleme aufgetaucht sind und wie wir diese gelöst haben wird im Folgenden diskutiert.

Authentifizierung Die Authentifizierung erfolgt für einen Benutzer mit den Logindaten des dort angemeldeten Benutzerkontos. Allerdings muss für den Aufruf über die REST-Schnittstelle für die jeweilige Applikation ein Key generiert und mitgeschickt werden. Die REST-Schnittstelle benutzt OAuth 1 für die Authentifizierung. Hierbei wird ein Session Token erstellt, das jeweils eine Gültigkeit von fünf Minuten besitzt. Damit nicht für jeden Aufruf eine eigene Authentifizierung erfolgen muss, wird ein Zeitstempel erstellt und vor dem Absetzen einer Anfrage überprüft. Die Authentifizierung wird von der privaten Methode `authorize` siehe Listing 7 durchgeführt. Der Parameter `signature` ist ein aus den restlichen Parametern berechneter Hashwert.

Quota Die REST-Schnittstelle unterstützt keine Möglichkeit, um den noch zur Verfügung stehenden Speicherplatz für den Account direkt abzufragen. Da das Interface diese Methode allerdings vorsieht, wurde hierfür eine private Hilfsfunktion `getUsedSpace`, in Listing 8 zu sehen, implementiert. Die Methode berechnet für einen als Paramter übergebenen Ordner (folderId) die Größe, indem Sie rekursiv alle Datei- und Ordnergrößen aufsummiert. Da die Größe des freien Speicherplatzes eine Konstante (zur Zeit 50 GB) darstellt, lässt sich der freie Speicher so leicht durch eine Differenz berechnen.

```
private void authorize() {
    Document responseDocument = null;
    final MultivaluedMap<String, String> queryParams =
      new MultivaluedMapImpl();
    queryParams.add("email", eMail);
    queryParams.add("password", password);
    queryParams.add("application_id", applicationId);
    queryParams.add("signature", signature);
    try {
      responseDocument =
    getMediafireResponse(AUTHORIZE_PATH, queryParams);
    } catch (final Exception e) {
      LOGGER.fatal(
    "Response from Mediafire: authorization failed.");
    }

    sessionToken =
      getTagValue(responseDocument, "response",
    "session_token")[0];
}
```

Listing 7: Authentifizierungsprozedur für Mediafire

Dateitransfer Das Hoch- beziehungsweise Herunterladen von Dateien ist bei Mediafire nicht so schön gestaltet worden wie z.B. bei Google Drive. Für das Downloaden von Dateien ist in der Mediafire-API ein Downloadlink vorgesehen, der für unsere Schnittstelle in einen Stream umgewandelt werden musste. Dazu benötigten wir lediglich eine Hilfsfunktion. Beim Upload von Dateien bekommt man einen `upload_key` zurück, der nicht wie intuitiv anzunehmen dem Schlüssel der Datei entspricht (`quick_key`), sondern als Parameter für den `poll_upload` Request dient. Damit lässt sich dann der Status des Uploads abfragen, welcher nach Beendigung den `quick_key` liefert. Um eine gewisse Effizienz zu erreichen, wird der Status des Uploads nach bestimmten Zeitintervallen durchgeführt.

6.4.5. 4shared Implementierung

4shared stellt für Entwickler zwei Schnittstellen zur Kommunikation bereit: eine REST- und eine SOAP-Schnittstelle.

REST-Schnittstelle Auf den ersten Blick schien die REST-Schnittstelle für unsere Zwecke geeignet. Sie unterstützt alle durch unser Interface benötigten Ressourcen:

- user - Informationen über den aktuellen User

```
private long getUsedSpace(final String folderId)
      throws StorageProviderException {
    long usedSpace = 0;
    final List<IMetaData> list = listFolder(folderId);
    if (list == null) {
      return usedSpace;
    }
    for (final IMetaData iMetaData : list) {
      if (iMetaData.isDirectory()) {
    usedSpace += getUsedSpace(iMetaData.getId());
      } else {
    usedSpace += iMetaData.getFileSize();
      }
    }
    return usedSpace;
}
```

Listing 8: Hilfsmethode getUsedSpace für Mediafire

- users - Accountinformationen beliebiger User
- folders - Ordnerinformationen
- files - Dateiinformationen

Außerdem stellt 4shared eine API-Konsole bereit, mit der Entwickler die REST-Aufrufe direkt testen können. Die REST-Schnittstelle benutzt OAuth 1 für die Authentifizierung. Allerdings war es uns nicht möglich, über die API-Konsole REST-Aufrufe mit Authentifizierung abzusetzen. Bei der Authentifizierungswahl OAuth 1 in der API-Konsole öffnete sich lediglich ein Fenster mit der Nachricht „Domain not allowed“. Aus Zeitgründen entschieden wir uns, erst die SOAP-Schnittstelle zu testen.

SOAP-Schnittstelle Bei dieser Schnittstelle handelt es sich um eine JAR-Datei zur Einbindung in eigene Java-Projekte, die direkt Methoden zur Kommunikation mit 4shared bereitstellt. Ein erster Test verlief erfolglos. Da allerdings schon sämtliche Methoden, die wir brauchten, in dieser Schnittstelle zur Verfügung standen und es mehrere Code-Beispiele gab, die diese Schnittstelle benutzten, entschieden wir uns die SOAP-Schnittstelle zu nutzen und die Fehler zu beseitigen. Der Quelltext der SOAP-Schnittstelle steht bei Google Code zum Download [1]. Dies ermöglichte es uns die Ursache der Exceptions zu finden. Es stellte sich heraus, dass die WSDL-Datei der SOAP-Schnittstelle aufgrund absoluter Pfade nicht gefunden werden konnte. So konnten wir die ebenfalls zum Download verfügbare WSDL-Datei auf

einem Server unter einer festen URL verfügbar machen und den Pfad im Quelltext der SOAP-Schnittstelle anpassen.

Nachdem die SOAP-Schnittstelle fehlerfrei arbeitete, konzentrierten wir uns darauf und sahen von weiteren Test der REST-Schnittstelle ab. Wir begannen mit der Implementierung der von uns erstellten Schnittstelle für 4shared auf Basis der SOAP-Schnittstelle. Auch die veralteten Java-Codebeispiele [2] zur Nutzung der SOAP-Schnittstelle ließen sich mit Hilfe des Quelltextes ohne großen Aufwand auf den neusten Stand bringen.

Die Implementierung der meisten Methoden unserer eigenen Schnittstelle mit Hilfe der SOAP-Schnittstelle von 4shared gestaltete sich einfach, es gab allerdings auch ein paar Problempunkte. So dauerte es beispielsweise lange, bis wir den Up- und Download von Dateien implementiert hatten. Unsere Schnittstelle sah die Verwendung von In- und Outputstreams vor.

6.5. Testen

Zum Testen der verschiedenen Implementierungen unserer Storage-Schnittstelle haben wir uns für JUnit [23] entschieden, da dieses Framework nicht nur in der Java-Welt weit verbreitet ist, sondern auch die meisten PG-Mitglieder Erfahrungen damit hatten. JUnit ist ein Framework, das Entwicklern dabei hilft Unit-Tests auszuführen, also einzelne Komponenten (z.B. Methoden) einer Anwendung oder Bibliothek zu testen. Dabei werden Methoden mit vorgegebenen Parametern aufgerufen und die Rückgaben mit den erwarteten Ergebnissen verglichen. Die Menge aller zu testenden Methoden, sowie deren Eingaben und erwartete Ausgaben nennt man *Testplan*.

6.5.1. Der Testplan

Der Testplan wurde in Gemeinschaftsarbeit erstellt. Zur Verwaltung des Testplans in schriftlicher Form haben wir unser Redmine genutzt. Alle Mitglieder der PG haben Testfälle (vor allem für die selbst verfassten Interface-Methoden) definiert. Da bei der Implementierung der Schnittstelle für die verschiedenen Storage-Anbieter teilweise unterschiedliche Rahmenbedingungen galten (z.B. verschiedene Restriktionen für Dateinamen) wurden die Testfälle von allen PG-Mitgliedern im Auge behalten und bei Bedarf geändert, oder erweitert. Es entstand der in Kapitel A.2 (Tabellen 6 bis 16) dargestellte Testplan.

6.5.2. JUnit Setup

Das Einbinden von JUnit war dank des Maven-Setups sehr einfach. Ein Eintrag in der `pom.xml` reicht aus und für eine einsetzbare Konfiguration.

Ein *Test Case* ist eine normale Java-Klasse, die Methoden mit der Annotation `@Test` enthält. Diese Annotation sorgt dafür, dass sämtliche damit annotierte Methoden von JUnit als Test ausgeführt werden, wenn man den *Test Case* startet.

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
</dependency>
```

Listing 9: Einbinden von JUnit per Maven

Die Besonderheit an unserem *Test Case* war, dass sämtliche Test-Methoden für alle von uns implementierten Storage-Anbieter ausgeführt werden mussten. Dies erreichten wir, indem wir die Test-Klasse folgendermaßen annotierten:

```
@RunWith(Parameterized.class)
```

Diese Annotation sorgt dafür, dass ein damit annotierter *Test Case* mit verschiedenen Parametern ausgeführt wird. Um diese Parameter zu definieren annotiert man eine beliebige statische Methode mit `@Parameters` und sorgt dafür, dass diese Methode eine `Collection` zurückgibt. JUnit erkennt dann beim Ausführen des *Test Cases*, dass dieser mit verschiedenen Parametern aufgerufen werden soll und startet diesen dann jeweils einmal mit jedem Parameter aus der `Collection`. Die Elemente der `Collection` werden dazu jeweils an den Konstruktor des *Test Cases* übergeben.

In unserem Fall erzeugten wir also eine `Collection` mit den verschiedenen Implementierungen unserer Storage-API (vgl. Listing 10).

```
@Parameters
public static Collection<Object[]> getParameters() {

    ...

    return Arrays.asList(new Object[][] {
        { new FourSharedStorage(...) },
        { new LocalStorage(...) },
        { new GoogleDriveStorage(...) },
        { new MediafireStorage(...) }
    });
}
```

Listing 10: Parametrisierung von Test Cases mit JUnit

Die Elemente dieser `Collection` wurden also jeweils an den Konstruktor des *Test Cases* übergeben.

```
private final IStorage storageImplementation;

public StorageTest(final IStorage storageImplementation) {
    this.storageImplementation = storageImplementation;
}
```

Listing 11: Konstruktor StorageTest anhand von IStorage

Die Klassenvariable `storageImplementation` (vgl. Listing 11) konnte daraufhin in jedem Test genutzt werden und enthielt je nach Ausführung die jeweilige Storage-Implementierung. Damit konnten nun sämtliche Storage-Implementierungen durch einen einzigen *Test Case* getestet werden.

Die Tests selbst wurden von den im Testplan angegebenen Mitgliedern der PG implementiert.

7. Demonstrationsprojekt Content-Battle

Um das entwickelte Konzept der Intercloud in der Praxis anwenden zu können, war es nötig, ein geeignetes Demonstrationsprojekt zu erstellen. Dieses Projekt soll im späteren Verlauf der Projektgruppe mit wenigen Handgriffen auf verschiedenen Konfigurationen (PaaS, Storage, DBMS) lauffähig sein und dort auch automatisch deployt werden können. Es handelt es sich bei dem Demonstrationsprojekt um eine Webseite, die auf einer Frontend/Backend-Architektur basiert. Dieses Kapitel beschreibt den Entwicklungsprozess der Webseite und beleuchtet die Teilbereiche der Konzeption des Testprojektes, der Entwicklung vom Backend und der vom Frontend.

7.1. Konzept

Das Demonstrationsprojekt stellt die erste lauffähige Version der Webseite und gleichzeitig den zu erreichenden Meilenstein für das erste Semester der Projektgruppe dar. Zu diesem Zweck musste sich zunächst auf das Thema der Webseite geeinigt werden, um im Anschluss daran Entscheidungen bezüglich Funktionsumfang, Architektur und Technologie treffen zu können.

7.1.1. Thema

Seit Beginn der Projektgruppe stand ein durch die Betreuer der PG vorgegebenes Thema für die Webseite im Raum: „Social Network für Bands“. Im Zuge der Planungen für diesen Meilenstein wurde von Miguel allerdings eine andere Möglichkeit für das Thema der Webseite vorgebracht und schließlich auch übernommen: das Content-Battle. Die grundsätzliche Idee ist, verschiedene Inhalte (Bilder, Videos, Text) in Turnieren gegeneinander antreten zu lassen und Nutzern eine Abstimmung darüber zu ermöglichen, welcher Inhalt der beste ist. Ausprägungen solcher Turniere wurden im Rahmen von Content-Battle Challenges genannt. An Challenges nehmen viele Nutzer mit ihren Inhalten teil. Das Thema einer Challenge wird durch ein Schlagwort festgelegt, an welchem sich dann die Inhalte orientieren müssen. Eine Challenge gliedert sich dabei in zwei grobe Phasen. Phase Eins ist eine Battle-Phase, bei der keine festen Paarungen existieren und Inhalte und deren Gegner zufällig ausgewählt und auf der Hauptseite angezeigt werden. Das Ziel dieser Phase ist lediglich, möglichst viele Benutzer für den eigenen Inhalt abstimmen zu lassen. Am Ende dieser Phase stehen die besten Inhalte, basierend auf der gesammelten Anzahl von Stimmen (Votes), fest. Diese Phase dient in erster Linie dazu, die qualitätiv höherwertigen Inhalte für die darauf folgenden KO-Phasen zu bestimmen. Würde beispielsweise direkt mit einer KO-Phase gestartet, könnten sich potenzielle Anwärter auf den Gesamtsieg bereits in Runde Eins begegnen, was ein hohes Frustpotenzial mit sich bringen würde. In der KO-Phase wird dann in Form eines Turnierbaumes in direkten Paarungen gegeneinander angetreten. Jede Runde dauert dabei eine gewisse Zeitspanne, an deren Ende ein Sieger feststeht, der eine Runde weiter ziehen darf, bis im Finale der Gesamtsieger ermittelt wird. Das Content-Battle kennt drei

verschiedene Rollen für die Benutzer:

- Gäste - Können lediglich für Inhalte abstimmen.
- Registrierte Nutzer - Können eigene Inhalte hochladen und an Challenges teilnehmen, sowie andere Benutzer zu Duellen fordern.
- Administratoren - Haben das Recht globale und Challenge-spezifische Parameter zu konfigurieren.

Für den weiteren Verlauf der Projektgruppe war zudem noch geplant, Nutzern für die Teilnahme an Challenges Entlohnungen in Form von Spielgeld zu geben, welches dann in Wetten auf Paarungen in den KO-Phasen vermehrt werden kann. Dies soll den Nutzern als zusätzliche Motivation dienen, mit eigenen Inhalten an Challenges teilzunehmen. Fernen waren Eins-zu-Eins Duelle geplant, bei denen Benutzer direkt und gezielt durch andere herausgefordert werden können.

7.1.2. Umfang Demonstrationsprojekt

Für den Meilenstein am Ende des ersten Semesters war lediglich ein Teil dieser Features angedacht und der Fokus lag darauf, eine funktionierende Basisversion der späteren Webseite zu erstellen. Der Funktionsumfang dieser Version beschränkt sich darauf, dass Benutzer (nach der Registrierung) die Möglichkeit haben, mit eigenen Inhalten an Battles teilzunehmen. Weiterhin sollte die Möglichkeit der Votes implementiert sein. Verbleibende Funktionen könnten in weiteren Arbeitsschritten geplant und realisiert werden.

7.2. Entwurf

Nachdem das grundsätzliche Thema und der Umfang der ersten Version der Webseite festgelegt wurden, mussten die daraus resultierenden Entscheidungen bezüglich der Umsetzung getroffen werden. Zum grundsätzlichen Konzept wurden zwei Alternativen diskutiert: Zum einen die Möglichkeit, statische Webseiten durch einen Webserver ausliefern zu lassen und zum anderen die Möglichkeit, ein JavaScript-Framework zu verwenden und im Hintergrund lediglich Anfragen an ein Backend, das REST-Services anbietet, zu richten. Zu diesem Zweck wurden einige Technologien diskutiert, die für diese verschiedenen Ansätze in Frage gekommen wären. Dabei handelte es sich um:

- Tapestry [37]
- play! [28]
- Spring [35]
- JavaServerFaces [31]

- JavaScript-Frameworks
    - [-] EmberJS [11]
    - [-] AngularJS [3]
    - [-] BackboneJS [6]

Alle Teilnehmer fanden die Idee interessant, ein JavaScript-basiertes Frontend zu erstellen, da die wenigsten damit bereits Erfahrung gesammelt hatten. Andererseits hatten fast alle Teilnehmer bereits mit Tapestry in einer ihrer Lehrveranstaltungen gearbeitet, sodass dieser Ansatz eine ernsthafte Alternative bildete. Die endgültige Entscheidung fiel auf die Lösung mit REST-Services und einem JavaScript-Frontend, da auf diese Weise dem Charakter der Projektgruppe als Lehrveranstaltung Rechnung getragen wurde. Dementsprechend wurde die Projektgruppe in Untergruppen geteilt, die sich mit Frontend und Backend befassten.

7.2.1. Backend

Das Backend beinhaltet die Persistenzschicht zum Halten von Daten in einer beliebigen Datenbank, die Kommunikationsschnittstelle für das Frontend zum Austausch von Daten sowie die interne Verwaltung von Daten zum Terminplanen (Scheduling) und zur Zugriffssicherung (Security). Die Entwurfsphase brachte folgende drei Kernelemente hervor.

Datenbankmodell Die in Abschnitt 7.1 beschriebenen Konzeptideen wurden für die erste Entwicklungsphase auf die in Abbildung 1 dargestellten Entitäten abgebildet. Die Klassenvariablen `SubmissionStart`, `SubmissionEnd`, `VoteStart`, `VoteEnd`, `KOStart`, `KOEnd` vom Type `Date` in der Entität `Battle` sind dabei für das Scheduling vorgesehen. Ziel ist es, den Server so zu konfigurieren, dass im Hintergrund ein zeitgesteuerter Mechanismus läuft, der automatisch die einzelnen Battle-Phasen (wie in Abschnitt 7.1 aufgeführt) startet bzw. beendet. Vorgesehen ist ebenfalls, dass einzig die Listen von Matchups für das Achtel-, Viertel und Halbfinale sowie der Gewinner eines Battles gespeichert werden. Bei Ende eines Battles sollen die irrelevanten Matchups entfernt werden. Auch diese Aufgabe soll durch das Scheduling realisiert werden.

Die Entität `Role` dient der Zugriffssicherung. Von der Realisierung über eine Enumeration wurde abgesehen, um eventuell Security Frameworks benutzen zu können und nicht von unserer individuellen Implementierung abhängig zu sein. Für die Lösung als Enumeration wäre ein selbstimplementierter Security-Lösungsansatz notwendig gewesen. Die Spezialisierung vom `User` zum `RegUser` impliziert bereits eine Rechtestruktur.

Die Entität `Matchup` ist der generische Typ von Begegnungen, der für alle Typen und Phasen von Battles verwendet wird. Spezielle Matchups, für die Entwicklungsphase 1 einzig die `KOMatchups`, realisieren dabei zusätzlich benötigte Attribute.

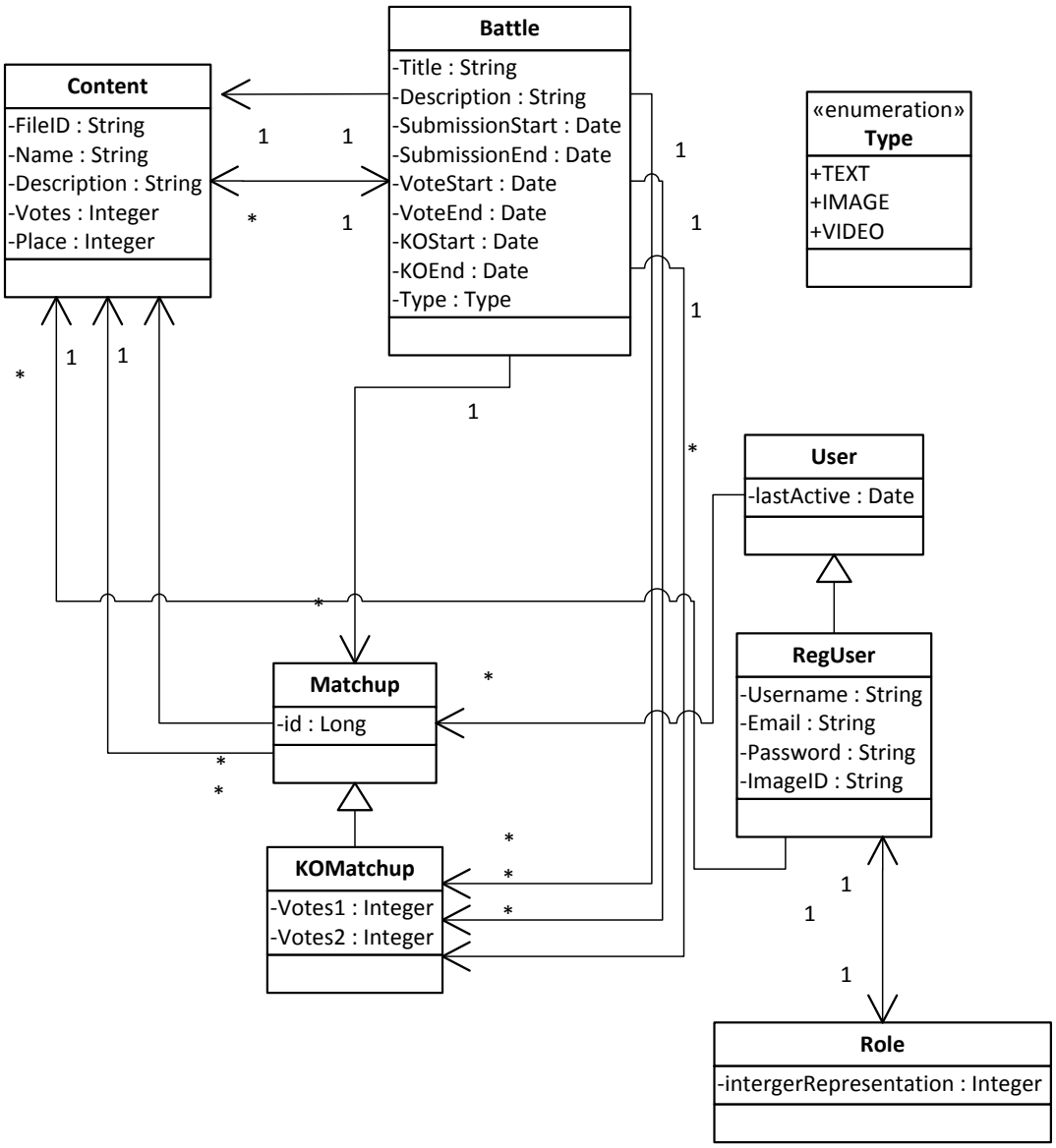

Abbildung 1: Content-Battle - Data Model

Content Entitäten verkörpern jede Art von Mediadatei im Demonstrationsprojekt. Sie halten die `fileID`, als Referenz zum Storage-Objekt, einige Zähler zu statistischen Zwecken sowie einen Verweis zu Battle und Matchup, an denen sie beteiligt sind. Dabei wurde vereinbart, dass aus Komplexitätsgründen jeder Content nur einem Battle zugeordnet wird. Benutzer müssen also für verschiedene Battles stets einen neuen Content erstellen (je eine Mediadatei hochladen), selbst wenn sie ein und dieselbe Datei an verschiedenen Battles teilnehmen lassen wollen. Das Löschen von Contents erwies sich ebenfalls als komplex, da viele Randfälle zu beachten sind (z.B. nimmt der Content an einem laufenden Battle in einer beliebigen Battlephase teil?). Deswegen wurde entschieden, dass Contents in der Datenbank gehalten wer-

den und das Löschen lediglich zum Verwaisen der Entität führt, `owner` sowie `fileID` (nach Löschen der Mediadatei auf dem Storage) werden also `null` gesetzt. So kann der Content nach wie vor vital an Battles teilnehmen, ohne gravierende Fehler zu erzeugen.

RESTful Webservices als Kommunikationsschnittstelle Als Schnittstelle zwischen Frontend und Backend wurden RESTful Webservices vereinbart.
Nach außen zugänglich gemacht werden die CRUD[3] Methoden auf den Entitäten, Methoden zum Abstimmen auf Contents/Matchups, Upload/Download von Mediadateien (nach Anbindung an die Storage-API, siehe Kapitel 6.4), Authentifizierung (Login, Logout) und Funktionen zum Starten/Stoppen des Scheduling (vgl. Anhang A.3).

Rollenkonzepte Die Zugriffskontrolle unterscheidet zwischen drei Rollen. Dem Administrator, dem registrierten Benutzer (RegUser) und dem unregistrierten Gast (User). Der Administrator hat alle verwaltenden Funktionen inne. Er kann Battles erstellen, bearbeiten und löschen, Verwaltungsroutinen steuern, Nutzerinhalte verändern sowie Benutzerkonten löschen. Die nächste Hierarchiestufe stellt der RegUser dar, der Content einstellen kann und seine Profilinformationen editieren darf. Dem User wird einzig gestattet, Matchups und zugehörige Contents in Battles anzusehen, zu bewerten und sich als RegUser zu registrieren. Diese Rollen stehen in Teilmengenbeziehungen bzgl. der Privilegien zueinander.

7.2.2. Frontend

Nachdem die Entscheidung, ein JavaScript-Frontend zu entwickeln, getroffen worden war, musste ein JavaScript-Framework ausgewählt werden. Die Entscheidung fiel auf AngularJS [3]. Den Ausschlag gab neben dem Funktionsumfang die verständliche, umfangreiche und einsteigerfreundliche Dokumentation.

AngularJS wurde in seiner ersten Version 2009 von Google unter der MIT Lizenz [24] veröffentlicht und wird, bis zum Zeitpunkt dieses Berichts, aktiv weiterentwickelt. AngularJS erweitert Browser-Anwendungen um das MVC-Konzept[4] und stellt weitere, AngularJS-spezifische Attribute für HTML-Tags (Directives) zur Verfügung, welche dann durch das Framework ausgewertet werden.
Nachdem der Entwurf durch das Backend-Team vorgestellt wurde, begann parallel die Arbeit am Frontend. Zu diesem Zweck wurde die Frontend-Gruppe weiter aufgeteilt. Drei Mitglieder wurden mit der Aufgabe der Programmierung des Frontends betraut (David, Sebastian F., Fabian), während die verbleibenden vier Mitglieder für den Designaspekt zuständig waren. Die erste Aufgabe der Design-Gruppe war es, Vorschläge für ein Design zu machen. Niklas und Dominik S. entwickelten zu

[3] Create Read Update Delete

[4] Model-View-Controller: Muster zur Strukturierung von Software-Komponenten in drei Teile - Datenmodell, Präsentation, Steuerung

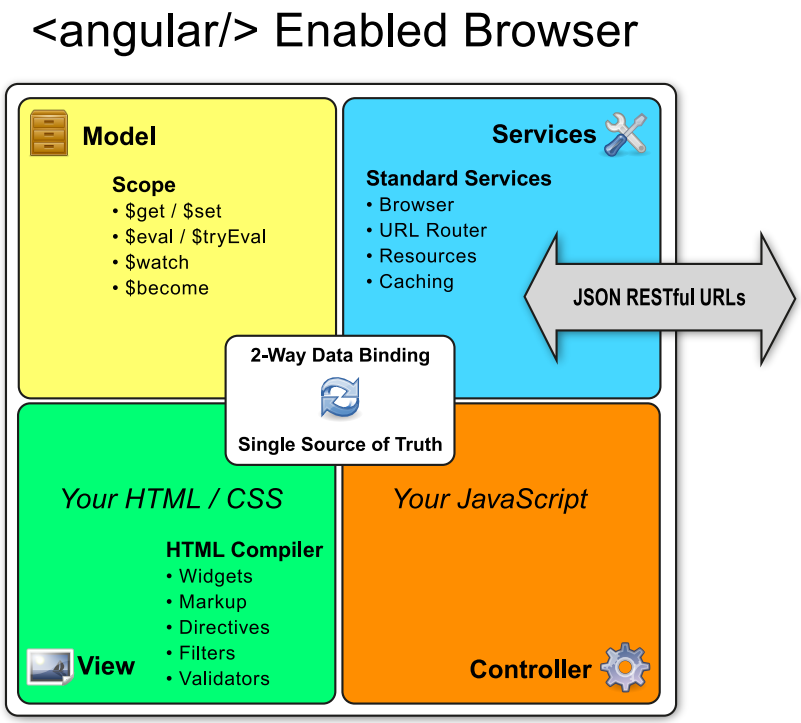

Abbildung 2: Prinzip Angular
Quelle: AngularJS - Offizielle Dokumentation [51]

diesem Zweck ein Design für die Webseite in Photoshop, Miguel und Jaouad bereiteten einen Vorschlag basierend auf Twitter Bootstrap[5] vor. Beide Vorschläge wurden bereits eine Woche später vorgestellt (Abbildung 3 und Abbildung 4). Während der Vorstellung beider Alternativen wurde beschlossen, eine Kombination zu verwenden und die Templates vorerst basierend auf Twitter Bootstrap zu erstellen, um sie dann später geeignet an das Look and Feel des Designs anzupassen. Zu diesem Zweck wurde eine Liste der benötigten Templates für einen ersten Prototypen ausgearbeitet:

- Login-Bereich, der auch später die Links zum Profil und Logout enthält
- Benutzer-Registrierung
- Battle-Erzeugung
- Liste aller Laufenden Battles - enthält auch gleichzeitig die Details für ein Battle und die Möglichkeit, mit eigenen Inhalten teilzunehmen
- Hauptseite - enthält zufällige Begegnungen, für die der Benutzer abstimmen kann

[5]Sammlung von Hilfsmitteln zur Gestaltung von Webseiten. Enthält HTML/CSS basierte Vorlagen für Standard-Controls auf Webseiten [12]

Im Verlauf des Semesters wurden einzelne Layout-Elemente mehrfach neu entworfen und diskutiert, ehe sie in das Twitter-Bootstrap Design eingebunden wurden. Die in Abb. 4 gezeigten Ideen wurden gänzlich verworfen. Der Meilenstein für das fertige Design wurde an das Ende des ersten Semesters gelegt, um mit der Umsetzung des Designs nicht den Meilenstein des lauffähigen Prototyps in Gefahr zu bringen.

Abbildung 3: Bootstrap Entwurf Frontend

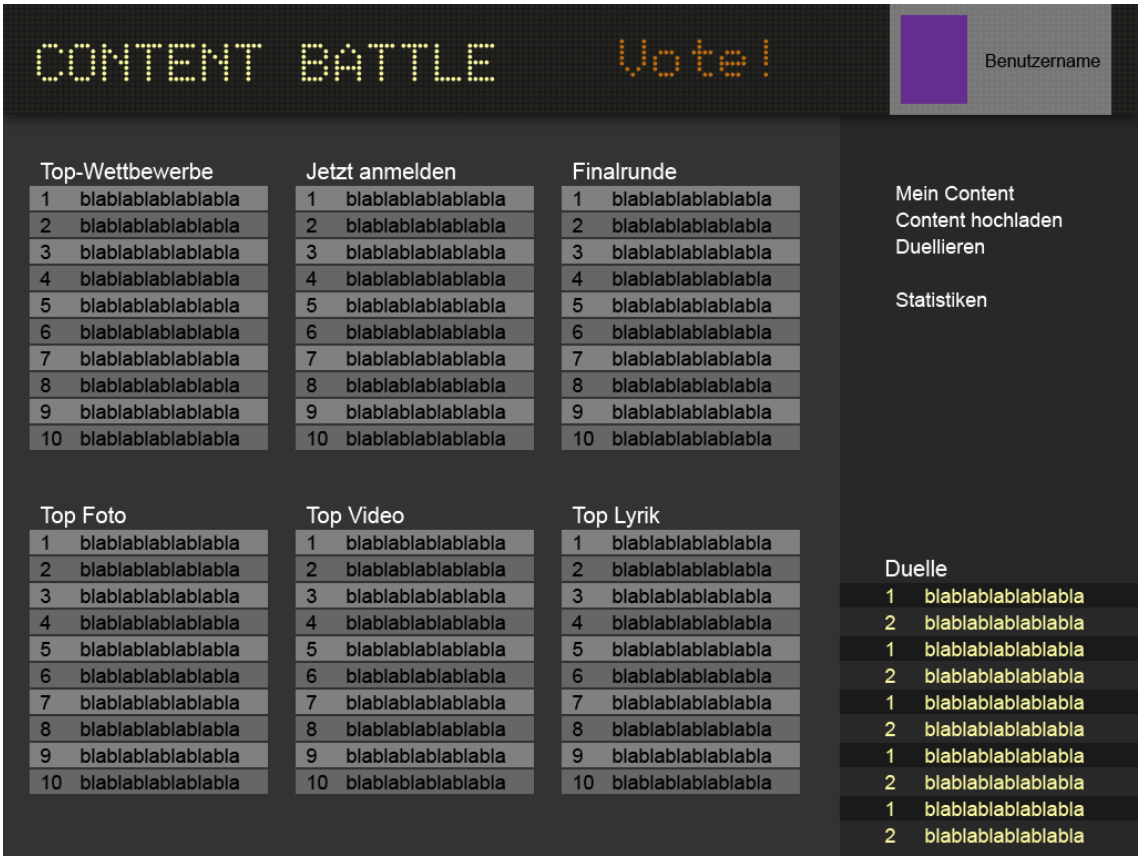

Abbildung 4: Photoshop Entwurf Frontend

7.3. Umsetzung

In der Entwurfsphase des Demonstrationsprojekts wurde prinzipiell die Entscheidung gefällt, die graphische Oberfläche (Frontend) vom Logik verarbeitenden Teil und der Persistenzschicht (Backend) klar zu trennen. Die dadurch erreichte Modularisierung hat verschiedene Vorteile, hauptsächlich aber erlaubt sie eine separate Entwicklung beider Komponenten, nachdem eine gemeinsame Schnittstelle vereinbart wurde. Die Details der Umsetzungen werden im Folgenden erläutert. Dabei arbeiteten Dominik Mohr, Sebastian Venier und Tilo Schulz am Backend, Miguel Cifuentes Perelló, Fabian Coerschulte, Sebastian Fechner, David Schoen, Dominik Sparer, Jaouad Zarouali und Niklas Zbick am Frontend.

7.3.1. Backend

Zur Umsetzung der Konzeptentwürfe der vorherigen Abschnitte kam das Spring Framework in Version 3.2.1.RELEASE zum Einsatz. Es bietet Möglichkeiten, die Entwicklung mit Java deutlich zu vereinfachen. Ein Ziel von Spring ist es, dem Entwickler möglichst viele Freiheiten bei der Umsetzung seiner Entwicklungsideen zu lassen, statt ihm ein starres Gerüst vorzugeben [74]. Man kann daher die einzelnen Teile des Frameworks unabhängig voneinander nutzen. Für die Entwicklungsziele sind die meisten Aspekte bereits in Springs Unterprojekten umgesetzt. Die folgenden Abschnitte beschreiben einige Spring-Pakete und deren Einbindung in das Backend.

Seit der Version 3.0 von Spring ist es möglich, konform zur Servlet-Spezifikation 3.0, Webanwendungen komplett im Quellcode zu konfigurieren. Die Notwendigkeit der XML-basierten Konfiguration entfällt. Diesen Weg schlägt das gesamte Backend-Projekt ein (mit Ausnahme eines kleinen Teilbereiches von Spring Security). Über die `@EnableMVC` Annotation signalisiert man der Anwendung, dass sie die Konfiguration aus dem Java Code holen soll (sog. JavaConfig). Mit `@Configuration` und `@ComponentScan` definiert man für die Konfiguration relevante Klassen. Über `@ImportResoure` lassen sich zusätzliche XML Konfigurationsdateien importieren. Und schließlich realisiert die eigentliche Servlet-Definition eine Klasse, die Springs Interface `WebApplicationInitializer` implementiert. In diesem Fall ist das die von uns erstellte Klasse `WebApplicationInitializer`.

Für den Betrieb des Backend Projektes wird ein Tomcat 7 benötigt. Da es sich hierbei um ein Maven Projekt handelt (vgl. Kapitel 6.4), ist der Start über den Maven-Befehl `tomcat7:run` mit dem Tomcat Plugin am einfachsten. Allerdings ist es auch möglich, das Projekt zu einem WAR-Archiv zu packen und auf einem beliebigen (aktuellen) Webcontainer zu installieren.

Spring JPA Das Paket Spring Data enthält komplette Unterstützungen für Persistenz und JPA Implementierungen. Für die Persistenz fiel die Entscheidung auf Hibernate, einen Object Relational Mapper der Laufzeitobjekte einer objektorientierten Programmiersprache (wie Java) direkt persistieren kann. Dementsprechend lag die Verwendung von Spring Data nahe.

Über dieses Paket lässt sich das vereinbarte Datenmodell (vgl. Abbildung 1) direkt und verlustfrei umsetzen. Dazu wurden codeseitig Java-Annotationen (statt einer `persistence.xml`) verwendet. Die mit `@Entity` annotierten Klassen stellen die Entitäten dar. Sie befinden sich im Package `de.tudortmund.cs.pg570.intercloud.backend.domain`.

Die Klasse `InfrastructureContextConfiguration` setzt die Datenbankverbindung mit den jeweiligen Verbindungsdaten über die mit JavaConfig definierte Bean `dataSource()`. Darüber lässt sich prinzipiell jede SQL-Datenbank anbinden. Für den lokalen Betrieb ist eine H2-Datenbank (inMemory-Datenbank, benötigt keine Installation und persistiert in den Hauptspeicher) voreingestellt. Sie wird gewählt, wenn keine anderen Verbindungsdaten vorliegen.

`@EnableJpaRepositories` wird benutzt, um Pakete anzugeben, in denen Klassen liegen, die zur Datenbankabfrage verwendet werden. Mit Spring Data werden CRUD-Methoden für Entitäten bereits automatisch generiert. Die Definition von eigenen Funktionen kann über die Definition der Methoden in Interfaces erfolgen. Um die eigentliche Implementierung kümmert sich das Spring Framework ebenfalls automatisch. So lassen sich beliebige Funktionen leicht definieren (vgl. Listing 12). Für komplexere Abfragen ist es auch möglich, mit der `@Query` Annotation die SQL-Abfrage direkt für eine Funktion zu definieren (vgl. Listing 12).

```
List<Content> findContentByOwner(RegUser owner);
List<Content> findContentByBattle(Battle battle);

@Query("SELECT u FROM User u WHERE TYPE(u) = User")
List<User> findAllGuests();
```

Listing 12: Spring JPA Methoden

Global wird für das Backend ein Transaktionsmanager verwendet, d.h. Änderungen an der Datenbank werden nur komplett eingespielt, wenn während der Verarbeitung kein Fehler auftrat, sonst wird ein Rollback ausgeführt und die Änderungen sind unwirksam. Um dieses Verhalten zu aktivieren, müssen Klassen/Methoden mit `@Transactional` annotiert werden.

Um das Problem der Kaskadierung von Objektbeziehungen zu vermeiden, bei dem mit einer einzigen Abfrage ganze Hierarchiebäume an den Benutzer ausgeliefert werden (sog. $n+1$-Selects-Problem aller Object Relational Mapper [56]), wird noch einmal zwischen internen und externen Entitäten unterschieden. Die externen Entitäten, die Data Transfer Objects (DTO), beinhalten statt tiefen Listen von referenzierten Objekten nur flache Listen von Objekt-IDs. Ferner enthalten sie keine sensiblen Daten wie Passwörter oder Email Adressen von registrierten Benutzern. Die DTOs dienen einzig der Kommunikation mit dem Frontend.

Problematisch dabei ist Hibernates `lazy loading`, ein Mechanismus, der Datenbankabfragen an Objektreferenzen erst sendet, wenn sie tatsächlich angefordert werden. Das ist ein sehr effizientes Verhalten, für die gewählte Umsetzung der DTOs allerdings ungeeignet. `Lazy loading` ist nur innerhalb einer Hibernate Session ak-

tiv, außerhalb der Session wirft der Zugriff auf nicht initialisierte Objektvariablen eine `LazyInitializationException`. Da der Datenbankzugriff erst erfolgt und nach Verarbeitung aus der resultierenden Entity ein passendes DTO erstellt wird, ist zu diesem Zeitpunkt die Hibernate Session bereits wieder geschlossen. Die Lösung für solche Probleme ist die manuelle Initialisierung der relevanten Variablen (sog. Hibernate Proxys) über die statische Methode `Hibernate.initialize(Object proxy)`. `LazyInitializationExceptions` werden so vermieden und die Entities lassen sich fehlerfrei in DTOs übersetzen.

Spring für JSON und Webservices Für die RESTful Webservices werden nach den Vereinbarungen (siehe Anhang A.3) Controller erstellt, die die Methoden realisieren und Aufrufe vom Frontend weiter an die Persistenzschicht leiten. Controller werden in Spring mit `@Controller` annotiert. Mit ihnen lassen sich URLs festlegen, auf denen die jeweiligen Funktionen lauschen. Dazu dient die Annotation `@RequestMapping(value="anyUri")`.

Wie bereits erwähnt, werden DTOs für den Datentransfer verwendet. Als Repräsentationsform wurde JSON gewählt. JSON ist ein schlankes, menschenlesbares Klartextformat, bei dem Objekte und deren Membervariablen als Key-Value-Paare kodiert werden. Spring bietet mit der Jackson-Bibliothek eine Möglichkeit, JSON automatisch zu en- und dekodieren. Dazu müssen im Controller lediglich die `produces` und `consumes` Parameter im `@RequestMapping` auf `MediaType.APPLICATION_JSON_VALUE` gesetzt werden. `Produces` legt fest, welches Datenformat nach außen geliefert wird und `consumes` signalisiert dem Service, dass im Request eine JSON `StringEntity` vorhanden ist, die zu einem Java Objekt umgebaut werden soll. Diese Translationsprozesse passieren automatisch im Hintergrund und bedürfen keines weiteren Eingreifens durch den Entwickler. Im Laufe der Implementierung stellte sich jedoch heraus, dass die Verwendung von DTOs teilweise problematisch ist, denn beim Anlegen und Verändern von Entitäten ist es notwendig, aus den in den DTOs gehaltenen IDs wieder Entitäten zu machen. Die einzige Lösung ist eine Rücktransformation, die jedoch viele Datenbankzugriffe benötigt.

Spring für Scheduling Die Terminplanung (Scheduling) betrifft im Rahmen dieses Projektes die zeitlich korrekte automatische Ansteuerung der einzelnen Phasen in einem Battle (Submission-, Random-, KO-Phase und Finished für abgeschlossene Battles). Darüber hinaus ist es notwendig, in Intervallen die Datenbank nach abgelaufenen Gastkonten zu durchsuchen und diese zu löschen. Während der Planung kamen zu diesem Thema mehrere Ideen auf:

- Cron Jobs
- Validierungsprozess immer bei einem Login laufen lassen
- Manuelle Ansteuerung durch den Administrator
- Spring Scheduling

Cron Jobs wurden dadurch ausgeschlossen, dass kaum einer der evaluierten PaaS Anbieter (bis auf Windows Azure) die Möglichkeit dazu bietet. Das Verhalten von Cron Jobs durch zeitgesteuerte Java Threads abzubilden war auch keine Alternative, da die meisten Anbieter das Erstellen zusätzlicher Threads aus Sicherheitsgründen untersagen. Die Aufgaben einem Administrator manuell zu überlassen, blieb nach gemeinsamen Beschluss eine denkbare, wenn auch nicht praktische Option. Auch sie wurde letztlich in Form von Servicecalls (siehe Anhang A.3) umgesetzt wurde. Das Ansteuern der Wartungsfunktionen bei jedem Login eines beliebigen Nutzers erschien zu kostenintensiv, da dies auch nicht asynchron vonstattengehen kann (Verbot von Threads). Spring Scheduling erwies sich damit als gute Alternative. Es deckt die gesetzten Anforderungen komplett ab. Während der Laufzeit der Webapplikation überwacht das Spring Framework im Hintergrund (beiläufig, durch den Webcontainer verwaltet) registrierte geplante Ereignisse und führt sie termingerecht aus. Dies kann zu einem festen Zeitpunkt oder in Intervallen geschehen. Zuständig hierfür ist die Klasse `MaintanceServiceImpl`. Über sie werden die Battlephasen überwacht und Wartungen/Aufräumarbeiten in zeitlichen Intervallen verwaltet. Einziger Nachteil dieses Werkzeuges ist, dass es nur codeseitig konfiguriert werden kann und nicht zur Laufzeit anpassbar ist. Man kann die eingestellten Intervalle oder Zeitpunkte also nicht nach Belieben im Nachhinein ändern. Da das Spring Framework aber schon verstärkt im Backend zum Einsatz kommt, wurde es auch für das Scheduling verwendet. In der Klasse `InfrastructureContextConfiguration` aktiviert `@EnableScheduling` das besagte Feature.

Spring Security Auch für die Zugriffskontrolle bietet Spring bereits ein etabliertes Projekt an: Spring Security. Es wird in Version 3.1.3.RELEASE verwendet. Das Projekt befindet sich zur Zeit der Erstellung dieses Berichtes in einer Umbauphase. Die Möglichkeiten der Servlet Spezifikation 3.0 werden erst nach und nach umgesetzt und Spring Security so erst schrittweise an die Weiterentwicklung vom Spring Framework angepasst. Die Unterstützung der JavaConfig ist noch nicht vollständig gegeben. Aus diesem Grund ist es notwendig, für die Benutzung von Spring Security eine Hybridlösung einzusetzen. Diese besteht aus der Verwendung von JavaConfig für die Beans und XML Konfiguration für die Zugriffskontrolle auf URL Pattern. Aber im Kern realisiert Spring Security alle Anforderungen, die in der Entwurfsphase an die Websicherheit definiert wurden.

Die Bean wird in `InfrastructureContextConfiguration` über `springSecurityFilterChain()` definiert. Dabei handelt es sich um die Default Filter Proxy von Spring Security. Sie realisiert den Standardfilter, der Anfragen an gesicherte URLs automatisch an den internen `UserDetailsService` weiterleitet. Damit die Nutzerkonten aus der Datenbank verwendet werden können, wurde ein eigener `UserDetailsService` namens `RepositoryBasedUserDetailsService` konzipiert. Diese Implementierung wird Spring Security über die `spring-security.xml` übergeben (vgl. Listing 14). Die Funktion `loadUserByUsername` gleicht dabei den Usernamen und danach das Passwort mit der Datenbank ab. Bei einer Übereinstimmung wird ein

gefülltes `UserDetails` Objekt zurückgegeben. Der Login war erfolgreich. Es ist nicht notwendig, für jeden Servicecall eine Art Accesstoken zu übergeben. Spring Security assoziiert die HTTP Session automatisch im Hintergrund mit einem `Principal` und speichert eine jSessionID ebenfalls automatisch in einem Cookie im anfragenden Browser. Prinzipiell ist eine Persistierung der SessionID auch möglich, um die Sicherheit zu erhöhen, jedoch übertrifft das die Anforderungen, die in der Entwurfsphase definiert wurden.

Um im Quellcode den aktuell angemeldeten Benutzer abzufragen, gibt es zwei Wege. Zum einen kann man schlicht eine Controllerfunktion um einen Funktionsparameter `Prinicpal principal` erweitern und Spring füllt bei jedem Servicecall diesen Parameter automatisch aus. Zum anderen kann man sich über `SecurityContextHolder.getContext().getAuthentication()` das passende `Authentification` Objekt holen, in dem Prinicpal und Zugangsdaten (Credentials) gehalten werden. Aus Sicherheitsgründen werden Passwörter nicht in Klartext gespeichert, sondern mit SHA verschlüsselt. Dieses Verhalten wird im Service bewirkt (vgl. Listing 13) und Spring Security über die XML Konfiguration mitgeteilt (vgl. Listing 14).

```
@Override
public RegUser addEntity(final RegUser entity) {
  entity.setId(null);
  // encode password with SHA
  entity.setPassword((new ShaPasswordEncoder())
    .encodePassword(entity.getPassword(), null));
  // reg user role, never create new admins
  entity.setRole(new Role(RoleUtil.ROLE_USER, entity));
  final RegUser ret = commonRepository
    .saveAndFlush(entity);
  Hibernate.initialize(ret.getHasVotedIn());
  Hibernate.initialize(ret.getContents());
  Hibernate.initialize(ret.getRole());
  return ret;
}
```

Listing 13: Password Encoder in addEntity Service zu RegUser

Die in der Entwurfsphase vereinbarten Rollen wurden umgesetzt zu `ROLE_USER` und `ROLE_ADMIN`. Wie sich URL Pattern gegen unbefugten Zugriff schützen lassen, zeigt exemplarisch Listing 15.

Gästen ist es gestattet, für Contents in Battles in der Random-Phase abzustimmen. Dabei wird anders als beim Abstimmen registrierter Nutzer nicht persisitiert, für welche Paarung ein Gast bereits abgestimmt hat. Potentiell lassen sich damit beliebig viele Stimmen für einen Content generieren. Unter anderem auch um diesen Effekt zu mildern, wurde das Zufallsverhalten für die Aufstellung der Paarungen implementiert. Nehmen hinreichend viele Contents an einem Battle teil, ist es aus-

```
<authentication-manager>
  <authentication-provider
    user-service-ref="repositoryBasedUserDetailsService">
    <password-encoder hash="sha"/>
  </authentication-provider>
</authentication-manager>
```

Listing 14: XML Konfiguration des Authentification Managers

```
<intercept-url pattern="/error"
    access="permitAll"/>
<intercept-url pattern="/denied"
    access="hasRole('ROLE_USER')"/>
<intercept-url pattern="/welcome"
    access="denyAll"/>
<intercept-url pattern="/users"
    access="hasRole('ROLE_ADMIN')"/>
```

Listing 15: Beispiele zur Sicherung von URL Pattern

reichend schwierig ein und dieselbe Zufallsbegegnung erneut zu schaffen. Generell ist es möglich einen impliziten Gastzugang zu erstellen. Spring Security bietet Lösungen dazu an. So lässt sich eine Anonymous Filter Chain neben der Default Filter Chain registrieren. Die individuelle Implementierung des betreffenden Interfaces würde es erlauben, beim Erstellen eines Gastzugangs bzw. dem Beginn eines HTTP Session als Gast (nicht angemeldeter Nutzer) gleichzeitig einen `User` in die Datenbank zu schreiben, so dass jeder Benutzer eine ID besitzt, egal ob registriert oder nicht. Diese ID ist essentiell für die Persistierung der Abstimmungen als Gast. Da wir jedoch keine Möglichkeit fanden, dem Frontend diese ID ohne einen entsprechenden HTTP Request zukommen zu lassen, entschieden wir uns gegen einen in der Datenbank gehaltenen Gastzugang. Da es sich bei Content-Battle vornehmlich um einen Prototypen zu Demonstrationszwecken handelt, ist der entstehende Mangel durch die Mehrfachabstimmung hinzunehmen.

Anbindung an die Storage API Die Unterstützung der Storage API erwies sich als simpel, da diese als JAR-Archiv konzipiert wurde und in einem Maven Repository zur Verfügung steht. In den Methoden zum Down- und Upload von Mediadaten wurde die API verwendet, ebenso beim Löschen von Contents und beim Löschen sowie Editieren von RegUser-Daten (Verbindung zum Profilbild als Mediadatei). Die Verwendung der API funktioniert nahezu fehlerfrei. Einzig mit dem Local Storage hat sich eine Fehlerquelle herausgestellt. Beim Download von Dateien müssen die Streams wieder geschlossen werden, sonst sind die verknüpften physischen Dateien schreibgeschützt durch permanenten Zugriff.

Fehlerbehandlung Zur Fehlerbehandlung wurde ein globaler Exception Handler verwendet. Spring bietet hierfür entsprechende Annotationen an. Der zu diesem Zweck implementierte `GlobalExceptionHandler` wird bei der Webapplikation mit `@ControllerAdvice` registriert und enthält fünf Beans (mit `@ExceptionHandler` annotiert). Jede dieser Beans kümmert sich um das Verarbeiten je einer bestimmten internen Ausnahme. Im Fehlerfall während einer Verarbeitung im Backend wird ein `StatusDTO` mit einer Fehlermeldung, einem internen Statuscode und dem HTTP Status Code 400 (Bad Request) zurückgegeben. Über die internen Statuscodes kann das Frontend leicht erkennen, welcher Fehler vorliegt. In jedem anderen Fehlerfall wird die Exception als `String` für die Fehlermeldung im `StatusDTO` zusammen mit dem HTTP Status Code 500 (Internal Server Error) zurückgegeben.

Ferner wurde ein eigener `CustomizableTraceInterceptor` erstellt, der es mittels AspectJ erlaubt, vorkonfigurierte Stellen im Quellcode zu überwachen (etwa bestimmte Methoden oder Klassen). Er zeichnet jeden Zugriff auf die in der `trace-context.xml` spezifizierten Codestellen auf und gibt über Log4J individuelle Meldungen aus. Dieses Werkzeug kann benutzt werden, um Kontroll- und Zugriffsflüsse im Backend nachzuverfolgen, die sonst nur schwer zu prüfen wären.

Unit- und Integrationstests Obwohl Spring selbst über eine Bibliothek zum Testen verfügt, entschieden wir uns hier jedoch für eine externe Bibliothek. Der Grund dafür ist, dass die Spring Test Suite auf Spring Webapplikationen ausgelegt ist und sie folglich nicht genau auf das Backend passt, welches einer anderen Idee in der Umsetzung folgt. Daher bedienen sich die Unit Tests der EasyMock Library. EasyMock ist ein Test Framework, mit dem vitale Umgebungen für isolierte Unittests simuliert werden können. Dieses Feature ist besonders hilfreich beim Testen der Services, da diese über zur Laufzeit instanziierte (Code Injection via `@AutoWired`) Repositories mit dem tieferen Backend interagieren. Die Unittests beschränken sich auf die Services, da alle anderen Mechanismen durch Spring weitgehend automatisch generiert werden und deren Tests durch das Framework selbst bereits abgedeckt sind. Die Codeabdeckung der erfolgreichen Tests pro Klasse liegt zwischen 86% und 100%, was für die Entwicklung eines Demonstrators nach Meinung der Gruppe ein hinreichend guter Wert ist.

7.3.2. Frontend

Während die Backend-Gruppe damit begann, das erstellte Datenmodell umzusetzen, konnte das Frontend-Team bereits mit der Erstellung der Views beginnen. Dabei handelte es sich, wie bereits angedeutet, um Templates, basierend auf Twitter Bootstrap, die durch Miguel und Jaouad umgesetzt wurden. Zeitgleich wurde ein Skelett der Webseite in AngularJS erstellt, welches lediglich Platzhalter als Views und statische Daten enthielt. Nachdem das Backend-Team eine erste lauffähige Version zur Verfügung gestellt hatte, konnte mit der Umsetzung der Models begonnen werden. AngularJS stellt dazu eine einfache, übersichtliche Möglichkeit zur Verfügung, um mit REST-Services zu interagieren. Die Resource Factory erzeugt unter Angabe einer

Adresse zu einem Service einen Wrapper um ein Objekt, der Methoden zur Kommunikation mit den Services (beispielsweise die post()-Methode) bereitstellt. Auf diese Weise wurden verschiedene Factory-Klassen erstellt, die es beispielsweise ermöglichen, User Objekte zu suchen oder zu kreieren, diese dann zu manipulieren und mittels des Wrappers mit einem entsprechenden Aufruf zu speichern. Das Listing 16 verdeutlicht dieses Prinzip basierend auf der erstellten RegUser-Factory, die ein User Objekt zurückliefert, das dann verändert und über den von Resource erzeugten Wrapper mittels `save` gespeichert werden kann. Ein Ausschnitt aus der zugehörigen RegUser-Factory findet sich in Listing 17 und demonstriert die Verwendung von Resource um ein RegUser-Objekt zu finden und dieses zurückzugeben.

```
var usr = RegUser.getById(1);
usr.username = 'test';
usr.$save();
```

Listing 16: JavaScript-Code Beispiel (Interaktion)

```
var User =$resource(serviceUrl + "reguser/:name/:uid",
  {uid: '@uid', name: '@name'},
     {
        'get': { method: 'GET' },
        'save': { method: 'POST' },
        'update': { method: 'PUT' },
        'delete': { method: 'DELETE' }
});

RegUser.prototype.getById = function(id, success, error) {
    var user = User.get({uid: id}, function() {
      success(user);
    }, function(data) {
      error(data);
});
```

Listing 17: JavaScript-Code Beispiel (Factory)

Während der Arbeit an den Models wurden die Templates fertiggestellt und in das Skelett der Webseite übernommen. Zu diesem Zeitpunkt mussten noch die Controller für die Webseite geschrieben werden. Die Implementierung der Controller beschränkte sich in den meisten Fällen auf einige Aufrufe der Factory-Methoden, um die durch den View dargestellten Inhalte zu finden, und diesen dann mittels Data-Binding in die Templates zu schreiben. Da für die erste Version der Webseite auch das Rechtesystem geplant war und Nutzer mit ihren Inhalten an Challenges teilnehmen können müssen, musste ein entsprechend persistierter Login möglich sein.

Im Rahmen der Entwicklung auf lokalen Systemen wurde eine Prozedur entworfen, die das entsprechende RegUser-Objekt in den LocalStorage schreibt und bei jedem Aufruf der Webseite überprüft, ob ein solches Objekt gespeichert wurde, folglich ein aktiver Login besteht. Dieses Konzept wird in der Produktivumgebung durch den Einsatz von Spring Security im Backend abgelöst. Abbildung 5 zeigt den Prototypen der Webseite.

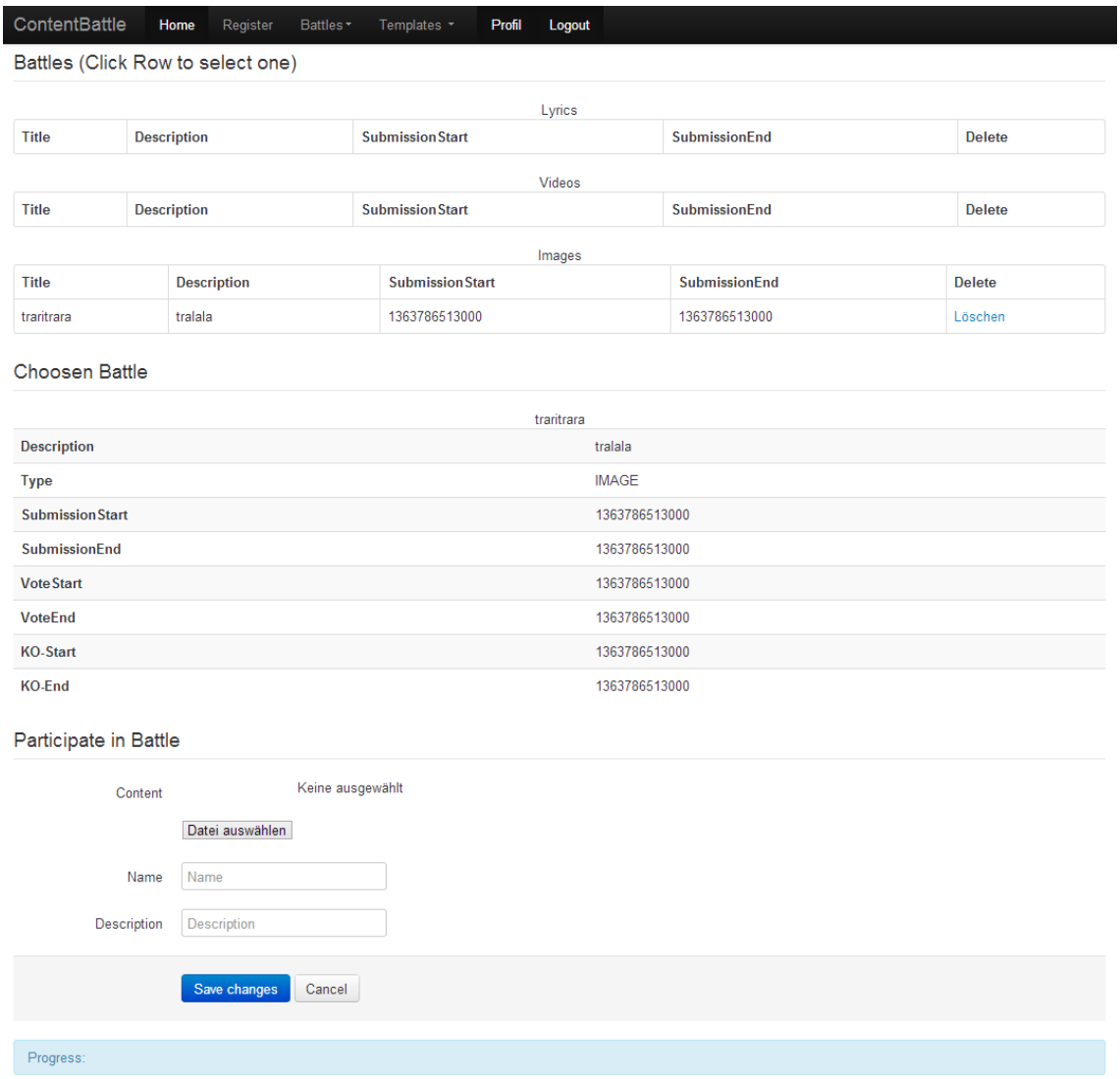

Abbildung 5: Prototyp Webseite

7.4. Abschluss des Demonstrationsprojektes

Mit dem Ende des ersten Semesters lag Content-Battle in einer lauffähigen Version vor. Die zuvor gesetzten Ziele wurden umgesetzt und alle Basisfunktionen wurden

implementiert. Die Gruppe entschied, den Demonstrator so zu belassen und keine weiteren Funktionen mehr umzusetzen. Als Prototyp beinhaltete diese Version Datenbankzugriff, Dateisystemzugriff (Storage) und eine Darstellungsschicht. Damit waren alle Bereiche abgedeckt, die die ursprünglich zu Beginn der Projektgruppe geplante Anwendung „Social Network für Bands“ auch beinhaltete. Statt auf die weitere Entwicklung an Content-Battle konzentrierte sich die Projektgruppe nach Erreichen dieses Meilensteines anschließend auf die Erstellung einer Installationsroutine um Applikationen bei verschiedenen PaaS-Anbietern deployen zu können sowie auf die Anwendungsmöglichkeiten der Storage-API in Fremdprojekten. Die ausstehenden Zusatzfunktionen (siehe Abschnitt 7.1.1) wurden verworfen.

8. Installer

Dieses Kapitel behandelt die Erstellung eines Installationsprogramms, im Folgenden „Installer“ genannt, durch die Projektgruppe, das Web-Anwendungen auf verschiedenen Clouds deployen kann. Zunächst wird ein Überblick über das Konzept und die generelle Idee des Installers gegeben, bevor im Anschluss die einzelnen Teilsysteme, Technologien und Probleme während der Entwicklung näher beleuchtet werden.

8.1. Konzeptionsphase

Ein zentraler Aspekt der Projektgruppe sollte es sein, eine Möglichkeit zu entwickeln, einzelne Komponenten einer Anwendung, die in einer Cloud eingesetzt wird, auszutauschen. Durch diese Entwicklung sollte es möglich sein, eine Anwendung beispielsweise auf einen anderen PaaS-Anbieter umzuziehen, oder gezielt einzelne Komponenten der Anwendung (z.B. Storage) auf eine andere Cloud auszulagern. Um diese Anforderung in einem geeigneten Rahmen bewerkstelligen zu können, wurde das Installer-Teilprojekt für das zweite Semester vorgesehen. Der fertige Installer sollte eine Oberfläche bieten, mit der ein Nuter die zu verwendenden Cloud-Services selbst auswählen kann, bevor im Anschluss der automatische Deploy-Prozess gestartet wird. Im Laufe der Konzeptionsphase wurde sich darauf geeinigt, nur ausgewählte Dienste zu unterstützen und den Installer lediglich für diese Teilmenge zu implementieren. Es handelte sich dabei um einen Machbarkeitsnachweis (Proof of Concept), da eine vollständige Abdeckung aller Cloud-Anbieter und Datenbank-Technologien den Rahmen der Projektgruppe gesprengt hätte.

Um die Dienste zu identifizieren, die durch den Installer unterstützt werden sollten, wurde eine Teilgruppe damit beauftragt, die im Rahmen der Seminarphase analysierten Anbieter auf das Vorhandensein von Schnittstellen für die Konfiguration und das Deployment hin zu untersuchen. Es wurden lediglich Anbieter in Betracht gezogen, die geeignete APIs oder Kommandozeilenwerkzeuge (command line interfaces, CLIs) für diesen Vorgang zur Verfügung stellen und nicht bereits in früheren Phasen der Projektgruppe verworfen wurden, da beispielsweise eine Kreditkarte für die Erstellung eines Kontos benötigt wurde. Tabelle 3 zeigt die Technologien, die während dieser Analyse ausgewählt wurden und durch den Installer unterstützt werden sollten.

Nachdem die geeigneten Technologien identifiziert wurden, musste sich auf eine geeignete Architektur für den Installer verständigt werden. Dazu wurden zunächst die folgenden Rahmenbedingungen für das Installer-Projekt abgesteckt:

- Unterstüzung von Unix- und Windows-Systemen
- Unkomplizierte Verwendung
- Einfache Einbindung der Cloud-Anbieter APIs bzw. CLIs

Typ	Anbieter
PAAS	Heroku CloudBees OpenShift
Storage	Lokal Google Drive 4Shared
DBMS	MySQL PostgreSQL H2

Tabelle 3: Liste der Installer-Technologien

- Möglichkeit der Ausführung in der Cloud[6]

Aufgrund dieser Rahmenbedingungen wurden drei Ansätze vorgestellt. Der erste Ansatz war eine Java-Anwendung, die beispielsweise durch eine auf Swing [36] basierende Oberfläche unterstützt werden könnte. Ein weiterer Ansatz war die Verwendung eines Installationsframeworks, wie zum Beispiel „Inno Setup“ [16]. Ein solches Framework könnte durch die korrekte Konfiguration ein Installationsprogramm für die entsprechenden Systeme erstellen. Der letzte Ansatz war das bereits bekannte Frontend/Backend-Konzept, welches bereits für das Demonstrationsprojekt verwendet wurde.

Durch eine anschließende Diskussion wurde sich darauf geeinigt, das etablierte Frontend/Backend-Konzept auch auf den Installer anzuwenden, da auf diese Weise die Möglichkeit besteht, den Installer als Web-Service zu betreiben, der durch ein leichtgewichtiges HTML-Frontend angesprochen wird. Durch diese Konzeption erhält man eine einfach anzupassende und wartbare Oberfläche, die auf allen aktuellen Plattformen angezeigt werden kann. Zudem kann sie durch einen Web-Server ausgeliefert werden. Weiterhin wurde entschieden, dass das Backend eine Funktionalität implementieren muss, die erkennt, welche Schnittstellen zu den Anbietern auf dem (Backend-)Rechner zur Verfügung stehen. Diese Informationen müssen durch das Frontend abrufbar sein, damit die Auswahlmöglichkeiten für den Nutzer dahingehend beschränkt werden können.

8.1.1. Backend

Nachdem festgelegt wurde, dass das Frontend/Backend Konzept für das Installer-Projekt verwendet wird, mussten auch hier die benötigten Spezifikationen getroffen werden. Um eine klar definierte Struktur innerhalb des Projektes aufzubauen, wurde

[6]Diese Rahmenbedingung wurde als optional angesehen.

die Verwendung des MVC Design Patterns[7] festgelegt. Des Weiteren wurde beschlossen, dass das JSON-Format (siehe Kapitel 3.2.6), wie im Backend, genutzt wird.

Als Grundgerüst für das Installer-Backend wurde eine Lösung gewählt, die sich bereits bewährt hatte: ein auf Spring basierendes Webserviceprojekt. Die Entscheidung wurde getroffen, da so bereits bekannte Lösungsansätze wiederverwendet werden konnten. Außerdem bot dieser Ansatz alles, was für den Installer benötigt wurde. Lediglich über das Verhalten dieses Grundgerüsts in Bezug auf die unterschiedlichen Systeme musste das Team sich noch Gedanken machen. Dabei stellte sich heraus, dass die Lösung grundsätzlich umsetzbar ist. Allerdings müssen für das Abfragen von vorhandenen Programmen und für das Ausführen einzelner Programme unterschiedliche Befehle ausgeführt werden. Dazu bietet Java jedoch die benötigten Mechanismen.

Dementsprechend wurde das vorgestellte Konzept zur Umsetzung freigegeben.

8.1.2. Frontend

Während der Konzeptionsphase einigte man sich innerhalb der Frontend-Gruppe darauf, ein HTML5-Frontend zu entwickeln und die Idee einer Swing basierten Oberfläche zu verwerfen. Die bereits angesprochenen Vorteile dieser Lösung sind, dass die Oberfläche auf allen Geräten und Plattformen angezeigt werden kann, die durch einen aktuellen Browser unterstützt werden. Weiterhin trug diese Entscheidung zur Flexibilität der Lösung bei, da das Frontend, neben der lokalen Anwendung, zusätzlich (optional) durch einen Web-Server ausgeliefert werden könnte. Die HTML5/JavaScript Kombination bietet zudem den nötigen Funktionsumfang, um die Oberfläche adäquat präsentieren und den Backend-Request erzeugen zu können, ohne dabei eine Entwicklungsumgebung und das Java-SDK vorauszusetzen. Durch die Einbindung von JQuery wurde die fehlende Funktionalität zum Absenden von Ajax-Requests hinzugefügt. Um eine fehlertolerante, übersichtliche und robuste Oberfläche zu schaffen, wurde ein Design-Entwurf erarbeitet. Dabei war es wichtig, dass der Nutzer zu jeder Zeit weiß, in welchem Schritt der Konfiguration er sich befindet und wie viele Schritte noch zu durchlaufen sind. Die Möglichkeit, zu jeder Zeit zu einem der früheren Schritte zurückzukehren um Eingaben zu ändern oder Fehleingaben zu korrigieren, war ebenfalls von Bedeutung. Abbildung 6 zeigt das Design des Installer-Frontends, welches auf Twitter Bootstrap, HTML5 und JavaScript basiert.

8.2. Entwicklungsphase

8.2.1. Backend

Nach der Konzeptionsphase mussten die nötigen Schnittstellen für das Frontend spezifiziert werden. Wir entschieden uns für ein minimalistisches Modell. So besitzt die

[7] Model-View-Controller

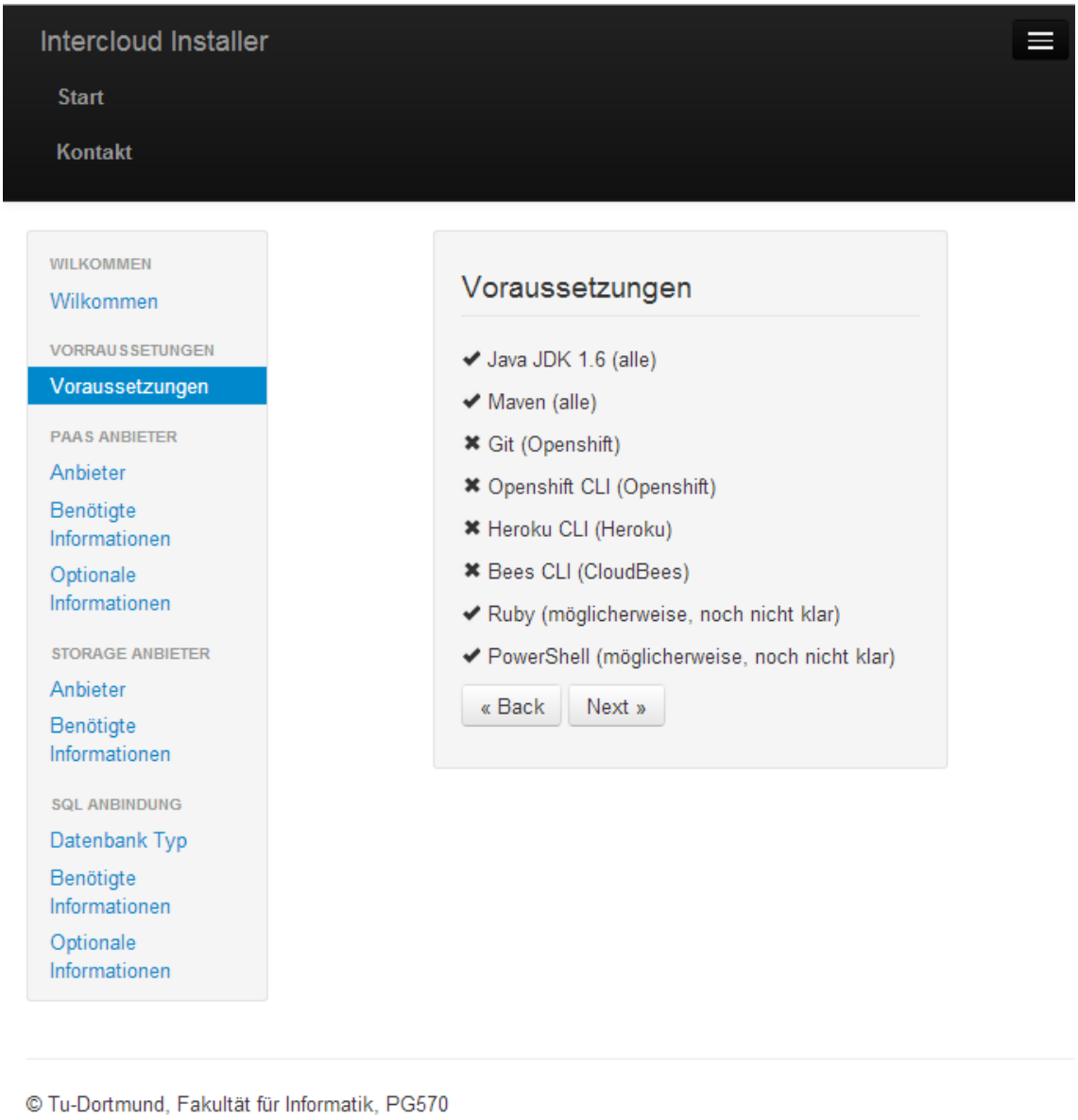

Abbildung 6: Design des Installer-Frontends

Schnittstelle zum Frontend lediglich zwei Services. Einen zur Prüfung der Installationsvoraussetzungen und einen zum Hochladen einer Webanwendung zum Dienst eines gewählten Anbieters. Wie schon im Demonstrationsprojekt wurde die Backend-Komponente mit dem Spring Framework entwickelt. Die implementierten RESTful Services finden sich nachfolgend in Tabelle 4. Der Kontrollfluss für die Interaktion mit dem Frontend sieht vor, dass zunächst die Voraussetzungen geprüft werden. Der Service `preconditions` stößt die Prüfung darauf an. Dabei werden alle in der Klasse `InstallerServiceImpl` enthaltenen Methoden mit dem Präfix „check“ ausgeführt. Sie setzen jeweils einen Konsolenbefehl für das jeweilige SDK ab (z.B. für Heroku

URL	HTTP Verb
/service/install/app	POST
/service/install/preconditions	GET

Tabelle 4: Webservices des Installer-Controllers

Toolbelt oder Screen). Wird der Aufruf erkannt, ist das jeweilige Werkzeug verfügbar. Dem Frontend wird dies über das ausgelieferte `PreconditionsDTO` mitgeteilt. Exemplarisch ist in Listing 18 die Methode `checkGit()` zum Prüfen der Installationsbedingung für Git aufgeführt. check-Methoden für weitere Programme ließen sich analog modellieren (vorausgesetzt sie unterstützen Kommandozeilenbefehle). Nachdem die Voraussetzungen geprüft wurden und für mindestens einen Anbieter

```
private void checkGit(final PreconditionDTO prec) {
 try {
   final StreamContainer streamContainer =
     CommandLineExecuter.execute("git version", null);
   final String execRet = streamContainer.out.toString();
   if (execResult.contains("git version")) {
     prec.setGitAvailable(true);
     prec.setGitVersion(execRet.substring(0,
       execRet.length() - 1));
   }
 } catch (final InstallerException e) {
   // nothing to do here
 }
}
```

Listing 18: checkGit() Methode zum Prüfen auf Installationsvorbedingung im Installer

alle benötigten Programme verfügbar sind (vgl. Abb. 6), wird die Installation bei einem Cloud-Anbieter angestoßen. Der nächste Schritt besteht im Aufruf des `app` Services. Dieser nimmt die im Frontend vorgenommenen Eingaben entgegen (sowohl textuelle als auch optionale Base64 enkodierte Dateien (vgl. Kapitel 8.3.1)) und verarbeitet sie.

Der Installer ist sowohl für beliebige (angepasste) Eingabeprojekte (vgl. Kapitel 9) als auch für das Demonstrationsprojekt „Content-Battle“ ausgelegt. Wird kein Drittprojekt durch das Frontend zur Verfügung gestellt, wird standardmäßig „Content-Battle“ benutzt. Zur Durchführung der Operationen auf dem jeweiligen Projekt wird Apache Maven benutzt. Zunächst wird entweder die im `HTTPRequest` übergebene ZIP-Datei oder die voreingestellte ZIP mit dem archivierten Demonstrationsprojekt entpackt. Im entpackten Projekt werden die Einstellungen modifiziert. Sämt-

liche Eingaben für die jeweiligen Cloud-Anbieter (wie Name des Storage-Anbieters oder Zugangsdaten) werden in Form einer Java Properties Datei übertragen. Diese Properties Datei wird direkt in das `/ressources/` Verzeichnis der entpackten Applikation kopiert. Schließlich wird der Befehl `maven package` aufgerufen und die so erstellte WAR-Datei zum Deployment im Verzeichnis `/warFile/` abgelegt. Der nächste Schritt besteht in der Generierung der Installationsskripte für die Anwendung.

Das Installationssetup unterstützt sowohl Unix/Linux als auch Windows-Plattformen. Die statischen Funktionen `isUnix()` und `isWindows()` der Klasse `CommandLineRunnable` dienen dem codeseitigen Feststellen des Betriebssystems. Notwendige Unix/Linux-spezifische Programme wie xte aus dem Paket XAutomation oder Screen werden nur unter dem betreffenden Betriebssystem geprüft. Die Laufzeitumgebung wirkt sich nicht nur auf die benötigten Programme aus, sondern auch auf die Art und Weise, wie Konsolenbefehle über das Backend ausgeführt werden. Dieses Verhalten wird in der Klasse `CommandLineRunnable` umgesetzt. Dazu werden die jeweiligen Ausführungsbefehle (execute commands) hart im Quellcode gehalten. Durch eine Weiche über das betreffende Betriebssystem wird für Windows die systemeigene `cmd.exe` und Linux/Unix eine `bash` aufgerufen. Anfänglich war auch die Unterstützung für Mac OS geplant. Davon wurde jedoch abgesehen, da die Projektgruppe keinen Zugriff auf ein passendes Testgerät hatte.

Die Ausführung der Konsolenbefehle stützt sich auf das Java-Paket `exec` der Apache Commons Bibliothek [9]. Das Paket dient der Steuerung und Ausführung von (System-) Prozessen. Benutzt werden die Klassen `DefaultExecutor` und `PumpStreamHandler`. Letztere kopiert die Standard- und die Fehlerausgabe beliebiger Subprozesse in die jeweilige Standardausgabe des dazugehörigen Elternprozesses. Die andere Klasse startet beliebige Kommandos über die Funktion `execute`.

Die Ausführung der Kommandos brachte speziell unter Windows unerwartete Probleme mit sich (siehe Kapitel 8.3.1). Um Befehle mit Nutzerinteraktion, d.h. mit benötigten Eingaben, nutzbar zu machen, werden sowohl unter Windows als auch Linux zur Laufzeit generierte Skripte benutzt.

Die Skripte für Windows verwenden die Windows Powershell [29]. Sie ist eine Adaption der traditionellen Unix-Shell, bekannt aus unixoiden Betriebssystemen. Die passenden Skriptdateien enden auf .ps1. Bei den unixbasierten Skripten handelt es sich um .sh Dateien.

Zur Generierung der Skripte wird das Interface `DeployerService` implementiert. Für jeden Anbieter existiert eine eigene Implementierung, die die Funktion `createScript()` umsetzt. Darin wird erneut auf das Betriebssystem geprüft und die jeweilige Routine angesteuert. Über einen `BufferedWriter` wird zeilenweise das Skript ins Dateisystem persistiert, um von dort aus ausführbar zu sein. Benötigte Eingabeparameter werden über eine `Map` im Konstruktor der Klasse übergeben. Die notwendigen Befehle der einzelnen SDKs wurden bereits in der Evaluationsphase der PaaS-Anbieter (siehe Kapitel 5) erarbeitet. Ein Beispiel zeigt Listing 19. Im Skript selbst wird eine an Visual Basic angelehnte Shell-Sprache benutzt. Über die Methode `SendWait` werden dabei Tasteneingaben simuliert. So wird das in Kapitel 8.3.1

geschilderte Problem umgangen. Für die Linux/Unix Skripte wird hauptsächlich das

```
final StringBuffer output = new StringBuffer();
output.append("[void] [System.Reflection.Assembly]::LoadWithPartialName
    (\"Microsoft.VisualBasic\")\n");
output.append("[Microsoft.VisualBasic.Interaction]::AppActivate(\"
    Powershell\")\n");
output.append("[void] [System.Reflection.Assembly]::LoadWithPartialName
    (\"System.Windows.Forms\")\n");
output.append("[System.Windows.Forms.SendKeys]::SendWait(\"heroku login
    ~\")\n");
output.append("[System.Windows.Forms.SendKeys]::SendWait(\"" +
    properties.get(USER) + "~\")\n");
output.append("[System.Windows.Forms.SendKeys]::SendWait(\"" +
    properties.get(PASSWORD) + "~\")\n");
output.append("[System.Windows.Forms.SendKeys]::SendWait(\"heroku
    plugins:install https://github.com/heroku/heroku-deploy~\")\n");
output.append("[System.Windows.Forms.SendKeys]::SendWait(\"heroku
    deploy:war--war " + properties.get(APPLICATION_NAME) + ".war" + "
    --app " + properties.get(APPLICATION_NAME) + "~\")\n");
```

Listing 19: StringBuffer zum Heroku Skript für Windows

Programm xte aus dem Paket XAutomation verwendet. xte simuliert Tasteneingaben auf dem Terminal, ähnlich wie die `SendWait` Methode unter Windows. Das Erstellen eines Shell-Skriptes ist in Listing 20 aufgeführt. Wichtig unter Linux/Unix Systemen ist die Rechtevergabe für die erstellte Skriptdatei. Diese muss für den Eigentümer ausführbar sein. Das wird durch den Zusatzbefehl `chmod u+x` erreicht. Da unter Linux/Unix das Skript sowie die darin komponierten Befehle zusammen direkt in der Shell ausgeführt werden, ist es nicht nötig, ein Zusatzwerkzeug wie die Powershell unter Windows zu starten. Somit entfällt auch ein zu `AppActivate` (vgl. Listing 19) äquivalenter Aufruf zum Fokussieren eines bestimmten Fensters bzw. eines bestimmten Programmes. Unter Windows ist ein solches Vorgehen notwendig, da die simulierten Tasteneingaben sonst nicht an das richtige Ziel geschickt werden können. In Kapitel 8.3.1 wird u.a. auch dieses Problem diskutiert.

8.2.2. Frontend

Um das während der Konzeptionsphase erarbeitete Design durch Kontrollfluss und Eingabemöglichkeiten zu erweitern, war es zunächst nötig, eine Liste darüber zu erstellen, welche Informationen für die einzelnen Cloud-Dienstleister nötig waren und somit durch den Request an das Backend geliefert werden mussten. Zu diesem Zeitpunkt hatte das Backend bereits die Analysephase der einzelnen Technologien beendet und die oben genannte Liste konnte auf Basis dieser Erkenntnisse erarbeitet werden. Auf dieser Grundlage wurde das Design zunächst um die nötigen Felder erweitert und der Kontrollfluss zwischen den einzelnen Abschnitten erstellt. Dabei war es wichtig, auf die durch den Nutzer getätigten Eingaben zu reagieren, da ein

```
final StringBuffer output = new StringBuffer();
output.append("#!/bin/bash\n");
output.append("xte 'keydown Control_L' 'key A' 'keyup Control_L' 'key C
    ' 'sleep 1'\n");
output.append("xte 'str heroku login' 'sleep 1' 'key Return' 'sleep 1'\
    n");
output.append("xte 'str " + properties.get(USER) + "' 'sleep 1' 'key
    Return' 'sleep 1'\n");
output.append("xte 'str " + properties.get(PASSWORD) + "' 'sleep 1' '
    key Return' 'sleep 1'\n");
output.append("xte 'str heroku plugins:install https://github.com/
    heroku/heroku-deploy' 'sleep 1' 'key Return' 'sleep 1'\n");
output.append("xte 'str heroku deploy:war --war "
  + folder.getAbsolutePath() + File.separator
  + properties.get(APPLICATION_NAME) + ".war --app "
  + properties.get(APPLICATION_NAME)
  + "' 'sleep 1' 'key Return' 'sleep 1'\n");
```

Listing 20: StringBuffer zum Heroku Skript für Linux/Unix

Nutzer, der beispielsweise den Anbieter Heroku gewählt hat, bei einem Klick auf „Benötigte Informationen“ auf die entsprechende, Heroku spezifische, Seite geleitet werden muss. Dazu wurde eine Anwendungslogik in JavaScript implementiert, die die Felder für die durch den Benutzer gewählten Technologien auswertet und basierend darauf die anzuzeigenden Bereiche einblendet. Das Ergebnis dieser Erweiterungen war ein Prototyp, dem lediglich die Logik für die Requests fehlte.

Um eine Anfrage zu erzeugen, die durch das Backend unterstützt wird, musste zu Beginn ermittelt werden, welche Plattformen das Backend unterstützt. Dazu wurde das DocumentReady-Event von HTML verwendet, um einen Request an das Backend zu schicken, welcher die installierten Voraussetzungen zurückliefert. Während der Bearbeitungszeit dieses Requests wird die Navigation des Frontends ausgeblendet und erst wieder zur Verfügung gestellt, sobald der Request bearbeitet und ausgewertet wurde. Listing 21 zeigt einen Ausschnitt des DocumentReady-Quellcodes, in dem zunächst die Schaltfläche „Next“ des Willkommen-Dialogs und die gesamte Navigation versteckt werden. Im Anschluss wird ein Request an das Backend erzeugt, um zu ermitteln welche Bedingungen erfüllt sind. Die Funktion „done“ wird aufgerufen, sobald der Request durch das Backend (asynchron) beantwortet wurde, und wertet die Antwort aus. Basierend darauf wird einerseits die Übersicht aus Abbildung 6 verändert, indem für alle erfüllten Bedingungen ein Haken gesetzt wird. Weiterhin wird der Inhalt der Auswahlboxen generiert, indem nur die unterstützten Inhalte aufgenommen werden. Die endgültige Anfrage wird durch das Betätigen der „Send“-Schaltfläche im letzten Schritt des Installers ausgelöst. Dazu werden alle für die Anfrage nötigen Informationen direkt aus den Feldern gelesen und in einem JavaScript-Objekt zusammengefügt, welches für die Anfrage in das JSON-Format konvertiert wird. Während der Konzeption wurde vorgeschlagen, den Installer auch für (beliebig geartete) Web-Anwendungen Dritter nutzbar zu machen. Ferner erfor-

```
$(document).ready(function () {
 document.getElementById("sbtn").style.visibility='hidden';
 document.getElementById("nlst").style.visibility='hidden';
 $.ajax({
  type: "GET",
  url: currenthost+'/intercloud-installer/installer/service/
     install/preconditions',
 }).done(function (msg) {
  precond = msg;
  if (precond.mavenAvailable)
   document.getElementById("pre-mvn").className="icon-ok";
  if (precond.gitAvailable)
   document.getElementById("pre-git").className="icon-ok";

  [...]

  var paas = document.getElementById('paasProvider');
  if (precond.herokuAvailable) {
   var option = document.createElement("option");
   option.text = "Heroku";
   option.value = "Heroku";
   paas.options.add(option);
  }

  [...]

  document.getElementById("sbtn").style.visibility='visible';
  document.getElementById("nlst").style.visibility='visible';

 }).fail(function (jqXHR, textStatus) {
  console.log("Request failed: " + textStatus);
  document.getElementById("sbtn").style.visibility='visible';
  document.getElementById("nlst").style.visibility='visible';
 });
});
```

Listing 21: JavaScript-Code DocumentReady

derte Google für seine Storage-Plattform GoogleDrive ein PKCS12-Zertifikat [8]. Aus diesen Gründen war es nötig einen Datei-Upload zu implementieren. HTML5 bietet, wie in [55] beschrieben, die sogenannte File-API an. Diese wurde verwendet, um die entsprechenden Dateien einzulesen. Die Dateiinhalte werden als Felder der Anfrage an das Backend übertragen, welches diesen Inhalt schließlich über einen `FileStream` in eine Datei schreibt. Listing 22 zeigt die Funktion `readFile`, welche die HTML5

[8] kurz: p12-Zertifikat - Public-Key Cryptography Standards

File-API verwendet, um eine Datei mittels `readAsDataURL()`[9] einzulesen und das Ergebnis asynchron über einen Callback an den Aufrufer zurückzusenden. Der untere Ausschnitt in Listing 22 zeigt die Verwendung der readFile-Methode und den anschließenden Request an das Backend. Die eingelesenen Daten werden zunächst in das Feld „zipFile“ geschrieben, bevor der Request erzeugt wird. Das Objekt wird mittels eines Aufrufs von `JSON.stringify` in das vom Backend erwartete JSON Format konvertiert.

```
function readFile(file, callback) {
 var reader = new FileReader();
 reader.onload = function (evt) {
 if (typeof callback == "function")
  callback(file, evt);
 };
 reader.readAsDataURL(file);
}

[...]
 readFile(zipfile, function (f, evt) {
  obj["zipFile"] = evt.target.result;
  //POST REQUEST$
  $.ajax({
   type: "POST",
   url: currenthost+'/intercloud-installer/installer/service/
      install/app',
   dataType: 'json',
   contentType: "application/json",
   async: true,
   data: JSON.stringify(obj)
  }).done(function (msg) {
   console.log(msg);
  }).fail(function (jqXHR, textStatus) {
   console.log("Request failed: " + textStatus);
  });
 });
[...]
```

Listing 22: JavaScript-Code DocumentReady

8.3. Probleme

Während der Entwicklungsphase traten in beiden Teilprojekten des Installers Probleme auf. Die Probleme im Frontend-Projekt waren, im Vergleich zum Backend-

[9]Diese Methode liest eine Datei ein, das Ergebnis ist Base64 kodiert

Projekt, deutlich geringer. Die einzelnen Probleme und angewendete Lösungsstrategien werden in diesem Kapitel erläutert.

8.3.1. Backend

Im Backend-Projekt traten drei größere Probleme auf. Das größte dieser drei betraf die Ausführung einzelner Befehle auf den verwendeten Systemen.

Kommandozeile Unter Windows sollte der Aufruf `cmd.exe /c + command` durch Javacode abgesetzt werden. Dabei steht `command` für den eigentlichen Befehl, der ausgeführt werden soll. Von dem so gestarteten Prozess wurde versucht die ein- und ausgehenden Logs zu verwerten. Die dazu verwendeten Methoden sind in Listing 24 dargestellt. Diese Implementierung brachte jedoch weitere Probleme zum Vorschein. Zum einen werden durch diesen Lösungsansatz einige Befehle als Unterprozesse der Kommandozeile gestartet. Sobald diese Unterprozesse Benutzereingaben erwarten, lassen sie sich programmatisch nicht durch Java eingeben. Zum anderen lassen sich die erzeugten Logausgaben dieser Unterprozesse nicht programmatisch durch Javacode auslesen. Dadurch entstand bei jeder Ausführung eine Blockade, die die Installation der gewünschten Anwendung verhinderte. Da sich dieses Problem durch Erproben verschiedener Lösungsstrategien nicht beheben ließ, wurde der skriptbasierte Ansatz gewählt, welcher bereits in Kapitel 8.2.1 dargestellt wurde. Das beschriebene Problem betraf nur die zur Installation benötigten Befehle, nicht jedoch die für die Abfrage der installierten Programme benötigten Kommandos.

Ein weiteres Problem in Bezug auf die Kommandozeile war der Befehl zum Ausführen eines erstellten Skriptes. Hier wurde unter Windows zunächst versucht, ein Skript mit dem Befehl `cmd.exe /c powershell & 'filename'` zu starten. Da dieser Befehl nicht zum gewünschten Ergebnis führte, wurden verschiedene weitere Befehle ausprobiert. In Listing 23 ist schließlich der funktionierende Befehl, um Skripte korrekt zu starten, aufgeführt.

```
SYSROOT\system32\WindowsPowerShell\v1.0\powershell.exe -
   noexit 'filename'"
```

Listing 23: Befehl zur Ausführung eines Powershell-Skipts

Unter Unixsystemen ist die Ausführung des erstellten Skriptes deutlich einfacher mit dem Befehl `screen filename`.

Dateiupload Sowohl bei der Entwicklung des Frontends, als auch bei der Entwicklung des Backends traten Probleme mit dem Hochladen einer Datei auf. Die Probleme im Frontend sowie deren Lösungen sind in Kapitel 8.3.2 beschrieben. In Bezug auf das Backend war zunächst angedacht, den Upload über eine eigene Methode im Controller zu realisieren. Dazu sollte die in Listing 25 dargestellte Methode verwendet werden.

Obwohl diese Methode funktionierte, musste ein anderer Weg gewählt werden. Der Grund dafür lag darin, dass die Installation einer Anwendung möglichst mit nur einer Anfrage realisiert werden sollte. Das bedeutet, dass sowohl die Konfiguration, als auch die für die Anwendung benötigten Daten innerhalb eines JSON-Objekts im Request übergeben werden musste. Um diesen Ansatz realisieren zu können, wurde zunächst serverseitig das Requestobjekt angepasst, um die benötigten Dateien annehmen zu können. Das in Kapitel 8.3.2 beschriebene Problem führte jedoch dazu, dass die Dateien nicht ordnungsgemäß übertragen wurden.

Der in Listing 26 gezeigt Code verarbeitet die mittels Base64 kodierte Datei und wandelt sie in ein Byte-Array um. Dieses Byte-Array kann im Folgenden mittels der Klasse `java.io.FileOutputStream` in ein `java.io.File` geschrieben werden.

Skripte Das Hauptproblem mit den Skripten ist, dass Befehle sowie die serverseitig benötigten Programme und Berechtigungen fest eingebundenen werden mussten.

Unter Windows muss, um ein Powershellskript ausführen zu können, zunächst der Befehl `Set-ExecutionPolicy RemoteSigned` ausgeführt werden. Ansonsten wird die Ausführung des erstellten Skriptes umgehend blockiert. Dieses Problem kann codeseitig nicht behoben werden und muss vor der Installation einer Anwendung ausgeführt werden.

Unter Unix muss eine Datei mit der Endung `.sh` die Berechtigung zur Ausführung haben, um codeseitig aufgerufen zu werden. Die benötigten Rechte werden hier mit dem Befehl `chmod u+x` vergeben (vgl. Kapitel 8.2.1).

Das Problem der fest einprogrammierten Skriptbefehle besteht weiterhin. Sollte sich auf der Seite der Cloudanbieter ein Befehl zur Installation von WAR-Archiven ändern, so wird auch das erstellte Skript nicht mehr funktionieren. Das lässt sich jedoch aus Sicht der Projektgruppe nicht ändern, da die Anbieter keine Schnittstelle zur Installation anbieten.

Letztlich besteht noch ein weiteres Problem in Bezug auf die Skripte, da sie die Eingaben des Benutzers simulieren. Sollte der Benutzer während der Ausführung des Skriptes in ein anderes Eingabefenster klicken, werden die Befehle an dieses Fenster gesendet. Hier gibt es, wie für das zuletzt beschriebene Problem, ebenfalls keine gangbare Lösung. Der Benutzer darf während der Ausführung nicht in ein anderes Eingabefenster klicken.

8.3.2. Frontend

Während der Frontend-Arbeiten trat lediglich ein Problem im Zusammenhang mit dem Datei-Upload auf. Dieses Problem resultierte darin, dass Dateien auf Frontend-Seite korrekt eingelesen wurden, im Backend nach dem Schreiben in eine Datei jedoch unbrauchbar waren. Es stellte sich heraus, dass für das Einlesen der Dateien die veraltete Methode `readAsBinaryString()` verwendet wurde. Diese Methode führt während des Einlesens eine Kodierung durch. Wie in [55] beschrieben, bietet die File-API Spezifikation zudem die `readAsDataURL()`-Methode, die als Rückgabewert

Base64 kodierte Daten liefert. Diese Kodierung besitzt die richtige Form für die Übertragung von Binärdaten.

```
static Thread createStartProcessInputHandlerThread(Process
   process) {
 Thread thread = new Thread(new Runnable() {
  @Override
  public void run() {
   PrintWriter printWriter = new PrintWriter(process.
      getOutputStream());
   Scanner scanner = new Scanner(System.in);
   try {
    while (scanner.hasNextLine()) {
     printWriter.println(scanner.nextLine());
     printWriter.flush();
     Thread.sleep(50);
    }
   } catch (InterruptedException interruptedException) {
    // ignore Process shutdown
   }
   scanner.close();
  }
 });
 thread.start();
 return thread;
}

static Thread createStartProcessOutputHandlerThread(Process
   process) {
 final Thread thread = new Thread(new Runnable() {
  @Override
   public void run() {
   try {
    char c = Character.MAX_VALUE;
    while ((c = (char) process.getInputStream().read()) !=
       Character.MAX_VALUE) {
     System.out.print(c);
    }
   } catch (IOException e) {
    e.printStackTrace();
   }
  }
 });
 thread.start();
 return thread;
}
```

Listing 24: Verwertung der ein- und ausgehenden Logs

```
@RequestMapping(value = "/form", method = RequestMethod.POST)
@ResponseBody
public UploadResponseDTO handleFormUpload(@RequestParam("file
   ") MultipartFile file) throws InstallerException {
 if (!file.isEmpty()) {
  installService.inputFile(file);
  return null;
 } else {
  return null;
 }
}
```

Listing 25: Controllermethode für File-Upload

```
String fileEncoded = "BASE64ENCODEDFILE";
byte[] file = null;
if (fileEncoded != null) {
 file = new Base64().decode(fileEncoded.substring(fileEncoded
    .indexOf("base64,") + 7, fileEncoded.length()));
}
```

Listing 26: Decode Prozess für Base64-kodierte Dateien

9. Externe Testprojekte

Parallel zur Fertigstellung des Installers arbeiteten drei PG-Mitglieder daran zu prüfen, ob die bisherigen Ergebnisse in Bezug auf Portierbarkeit und Vielseitigkeit auch auf andere Softwareprojekte angewendet werden können. Konkret bedeutete dies, dass sich jeder einem frei zugänglichen Open-Source-Projekt widmete und versuchte, die entwickelte Storage-API (siehe Kapitel 6) in dieses Projekt in integrieren. Die Idee dahinter war, zu evaluieren, wie viel Aufwand es erfordert, die Storage-API in bereits bestehender fremder Software einzusetzen.

Die erste Herausforderung bestand darin, geeignete Projekte zu finden. Manche Projekte erfordern aufgrund ihrer Komplexität umfangreichere Maßnahmen als andere, wie wir bei der Untersuchung des eCommerce Projektes Broadleaf Commerce [7] feststellen mussten. Dieses wirkte zwar auf den ersten Blick geeignet, da es auf Spring [35] und Maven [57] basiert, schied jedoch wegen des Codeumfangs aus.

Die Wahl fiel schließlich auf jTrac (Dominik S.), Apache Roller (Sebastian F.) und jforum (Niklas). Diese Projekte, die jeweiligen Maßnahmen, die für die Verwendung der Storage-API getroffen werden mussten, sowie die dabei aufgetretenen Probleme werden im weiteren Verlauf dieses Kapitels genauer beschrieben.

Darüber hinaus sollte ursprünglich auch untersucht werden, inwieweit diese Projekte mit dem von der Gruppe parallel entwickelten Installer funktionieren könnten. Aus Zeitgründen wurde letztlich jedoch von diesem Ziel abgesehen.

9.1. jTrac

Als erste Testanwendung für eine Integration der Storage-API wurde jTrac gewählt. jTrac ist eine auf Java basierte Issuetracking Web-Applikation. Im Folgenden wird die Anwendung kurz umrissen, wobei Installation und Struktur im Vordergrund stehen. Darauf aufbauend wird der Quellcode näher betrachtet, wobei aufgetretene Probleme bei der Integration aufgezeigt werden.

9.1.1. Download, Deployment und Installation

Bei jTrac handelt es sich um ein Open Source Projekt, das unter der Apache Lizenz [4] veröffentlicht ist. Somit ist eine Verwendung und Abänderung im Rahmen der Projektgruppe unbedenklich, da unter Apache Lizenz stehende Software frei geändert werden darf, so lange die Apache Lizenz und ein Verweis auf die Originalsoftware beigelegt wird.

Bevor mit der Integration der Storage-API begonnen werden kann, müssen einige Vorbereitungen getroffen werden. Die Java Quellen können über die offizielle jTrac-Webseite [22] heruntergeladen werden. Um den Quellcode auf einem Java Webserver ausführen zu können, muss dieser zunächst kompiliert werden. Hierfür wurde wie bereits bei den bisherigen Programmieraufgaben die Kombination aus Eclipse IDE und dem Webserver Tomcat gewählt. Des Weiteren wurde eine MySql-Datenbank vorbereitet. JTrac bedient sich hierbei des Hibernate Frameworks um die

Datenbankzugriffe auszuführen. Die zugehörige Konfigurationsdatei für die Hibernate Verbindung kann unter `src/main/resources/jtrac.hbm.xml` gefunden und verändert werden.

Alle zur Ausführung notwendigen Abhängigkeiten werden mit Hilfe des Build Management Tools Maven koordiniert. Bevor jTrac erstellt wird, sollten die beiliegenden Properties-Dateien angepasst werden. Konfigurationsmöglichkeiten die von jTrac angeboten werden, können in den Dateien `src/main/resources/jtrac-init.properties` und `src/main/resources/messages.properties` vorgenommen werden.

9.1.2. Einbau der Storage-API

Um die Storage-API in das jTrac-Projekt integrieren zu können, musste zuerst die von der Projektgruppe erstellte Storage-Bibliothek mit in das Projekt eingebunden werden. Hierfür muss in der Datei `pom.xml` eine neue Abhängigkeit hinzugefügt werden. Diese ist in Listing 27 aufgeführt. Um weitere Konfigurationsmöglichkeiten zu

```
<dependency>
    <groupId>intercloud</groupId>
    <artifactId>intercloud-storage</artifactId>
    <version>0.0.2-SNAPSHOT</version>
</dependency>
```

Listing 27: Maven Dependency für das Storage Teilprojekt

schaffen, wurde eine neue Properties-Datei `src/main/resources/Storage.properties` angelegt. Diese enthält alle wichtigen Informationen, die für die Benutzung der Storage-Bibliothek notwendig sind.

Um eine freie Auswahl der Storage-Anbieter zu gewährleisten, wurde jTrac die Klasse `StorageFactory.java` hinzugefügt. Sie kam bereits im Backend von ContentBattle (Kapitel 7) zum Einsatz und wurde an dieser Stelle nur wiederverwendet. Sie beinhaltet einen Konstruktor, der je nach Konfiguration die benötigten Anmeldeinformationen lädt und eine neu erstellte Methode `createFile(Attachement attachement, File file)`, die die Datei auf den gewünschten Storage-Anbieter hoch lädt (siehe Listing 28). Dabei sind im Objekt `Attachement` alle Meta-Daten gespeichert und das Objekt `File` enthält den dazugehörigen Datenstrom.

Im nächsten Schritt mussten die kritischen Punkte im Quellcode identifiziert werden, an denen die Storage-Bibliothek eingebunden werden soll. Dies gestaltete sich aufgrund der unübersichtlichen Struktur und nicht vorhandener Interfaces schwierig.

Die von den jTrac-Entwicklern vorgesehene Sicherung von Anhängen ist, alle wichtigen Informationen, wie Meta-Daten, ein zugehöriges Ticket etc., im bereits angesprochene Java-Objekt `Attachement` zu speichern und zu serialisieren. Zusätzlich wird das Dokument auf der lokalen Festplatte gesichert und ein zugehöriger Datensatz in der MySql-Datenbank angelegt. Diese Aktionen sind dezentral im Quellcode an die Stellen verteilt, an denen sie benötigt werden.

```
public void createFile(Attachment p_attachment, File file)
throws Exception{ if (m_provider.equals("GOOGLEDRIVE")) { try
    {
  m_metadata = new GoogleDriveMetaData();
     }catch(Exception ex){
                System.out.println(ex.getMessage());
            }
 } else if (m_provider.equals("FOURSHARED")) {
     m_metadata = new FourSharedMetaData(p_attachment.
        getFileName(), false);
 } else if (m_provider.equals("LOCAL")) {
     m_metadata = new LocalMetaData();
 } else if (m_provider.equals("MEDIAFIRE")) {
     m_metadata = new MediafireMetaData();
 }
    double size = file.length();
    m_metadata.setFileName(p_attachment.getFileName());
    m_metadata.setId(Long.toString(p_attachment.getId()));
    m_metadata.setFileSize(file.length());
    m_metadata.setMimeType("text/plain");
    InputStream fis = new FileInputStream(file);

    m_storage.uploadFile(fis, m_metadata, m_config.
       getProperty("parentid"));
}
```

Listing 28: Methode createFile

Die Storage-API wurde in diesem Testprojekt für das Speichern von Anlagen, zu den dazugehörigen Tickets, verwendet. Um das Anfügen von Anlagen vom Dateisystem auf die Storage-Anbieter umzuleiten, mussten in der Klasse `info.jtrac.JtracImpl` diverse Methoden, die zum Speichern bzw. Laden von Dateien genutzt werden, angepasst werden. Als Beispiel wurde in der Methode `writeToFile(FileUpload fileUpload, Attachment attachment)` das Speichern von Anhängen auf die Storage-API umgeleitet (vgl. Listing 29).

Schwierigkeiten Nicht alle PaaS-Anbieter unterstützen das Ablegen von Dokumenten auf deren Servern. Vor allem bei kostenlosen Anbietern, die in der PG hauptsächlich zum Einsatz kommen, wird dies häufig nicht erlaubt. Dies führte dazu, dass jTrac interne Fehler auf den Cloud-Plattformen hervorrief, auf denen es zu Testzwecken ausgeführt wurde. Daher musste der gesamte Quellcode auf etwaige Dateisystemzugriffe untersucht werden. Um dabei die Funktionalität nicht einschränken zu müssen, wären Workarounds benötigt worden.

Durch den unübersichtlichen Aufbau des Quellcodes und den Verzicht der Ent-

```
writeToFile(FileUpload fileUpload, Attachment attachment) {
   if(fileUpload == null) { return;
    }
    File file = AttachmentUtils.getFile(attachment, jtracHome
       );
    try {
        fileUpload.writeTo(file);
        m_factory.createFile(attachment, file);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }

}
```

Listing 29: Methode writeToFile über Storage-API

wickler auf Interfaces, war die Suche nach den Stellen im Quellcode, an denen Veränderungen vorgenommen werden müssen, sehr mühsam. Außerdem wurde dadurch erschwert, einen Überblick über die Applikation zu erhalten.

9.1.3. Fazit

Es war möglich, die Storage-API zumindest teilweise in das Projekt jTrac einzubinden und auch zu nutzen. Zu den Teilbereichen, die Eingebunden werden konnten, zählten die zentralen Funktionen zum hoch- bzw. herunterladen von Dateien und den damit verbundenen Metadaten auf die Storage-Server. Durch die Verteilung der dafür notwendigen Funktionen, mussten in vielen Klassen Änderungen vorgenommen werden. Es konnten allerdings nicht alle lokalen Dateisystem zugriffe unterbunden werden, die bereits im vorherigen Abschnitt SSchwierigkeitenërwähnt wurden. Eine derartiger Eingriff hätte zu großen Quellcodeänderungen geführt, dabei hätten komplette Klassen bzw. Funktionen neu programmiert werden müssen. Daher konnte das so umgeschriebene Projekt jTrac nur auf lokalen Rechnern ausgeführt werden und nicht auf den von uns genutzten kostenlosen PaaS-Anbietern.

9.2. Apache Roller

Als zweite Anwendung für die experimentelle Einbindung der Storage-API wurde Apache Roller [5] gewählt. Hierbei handelt es sich um ein quelloffenes Blog Server System unter der Apache Lizenz. Es ermöglicht die Erstellung und Verwaltung von Blogs und Benutzern. Große Firmen, die Apache Roller verwenden, sind unter anderem Oracle [26] und IBM [15].

Die Software benötigt für die Ausführung einen Java-EE-Server wie Tomcat sowie die Anbindung einer relationalen Datenbank mit Unterstützung für UTF-8 wie PostgreSQL [39]. Durch die Voraussetzungen zum Betrieb von Apache Roller sowie

der durch die Lizenz gestatteten Möglichkeit zur Anpassung des Quellcodes ist das Projekt gut für die Einbindung der Storage-API geeignet.

9.2.1. Download, Deployment und Installation

Für Entwickler wird der Quellcode über Apache Subversion (SVN) bereit gestellt [30]. Dadurch kann der Code über SVN heruntergeladen werden. In einem ersten Schritt muss Apache Roller nach dem Download für den eigenen Betrieb über eine Konfigurationsdatei angepasst werden. In der Datei `roller-custom.properties` können unter anderem Angaben und Einstellungen zum Installationsmodus, der Datenbankanbindung und der Speicherung von Dateien durch Apache Roller gemacht werden. Durch Einträge in dieser Datei werden Standardeinträge der Datei `roller.properties` im Verzeichnis `src/main/resources/org/apache/roller/weblogger/config/` überschrieben. In der zuletzt genannten Datei können also mögliche Einstellungen eingesehen werden. Notwendig für einen erfolgreichen Start der Software sind lediglich die Angaben zur Datenbankanbindung. Allerdings kann ein Eintrag für die automatische Installation über `installation.type=auto` unnötige Arbeit bei der Einrichtung ersparen. Dadurch werden beim ersten Start des Servers automatisch die nötigen Relationen der Datenbank erzeugt. Das manuelle Ausführen von SQL-Skripten wird einem so abgenommen. Für die Datenbankanbindung finden sich in der Standard-Datei die Einträge:

```
database.jdbc.driverClass=
database.jdbc.connectionURL=
database.jdbc.username=
database.jdbc.password=
```

Für eine notwendige Anbindung müssen also sowohl die Java-Klasse der Datenbankschnittstelle, die URL der Datenbank als auch Benutzername und Passwort für den Zugriff angegeben werden. Die für die Integration der Storage-API wurde eine PostgreSQL Datenbank verwendet.

Das Build-Management von Apache Roller ist durch Apache Maven realisiert. Über das Ausführen des Maven-Befehls `mvn clean install` kann der Build-Zyklus für die Software durchlaufen werden. Neben dem Kompilieren werden so die bereits enthaltenen Tests durchgeführt und bei unveränderter Konfiguration von Maven per `pom.xml` eine WAR-Datei erzeugt. Diese Datei enthält nun eine lauffähige Version von Apache Roller und kann mithilfe eines entsprechenden Webservers ausgeführt werden.

Beim Deployment des fertigen WAR-Archives fiel auf, dass ein Start der Software nur möglich war, wenn im Bibliotheksverzeichnis des verwendeten Tomcat-Servers Bibliotheken für das JavaBeans Activation Framework (JAF), JavaMail und die verwendete Datenbankschnittstelle Java Database Connectivity (JDBC) vorlagen. Während der JDBC-Treiber natürlich benötigt wird, sind im Gegensatz dazu in der Dokumentation von Apache Roller keine Angaben zur Notwendigkeit der anderen beiden Bibliotheken gemacht worden. Hingewiesen sei an dieser Stelle darauf, dass

die JavaMail-Bibliothek auch dann benötigt wird, wenn mögliche Funktionen über Mail Server nicht genutzt und dadurch auch erst gar nicht konfiguriert werden.

Nach dem Start des Webservers kann über ein Webinterface die Installation von Apache Roller, die im Fall der automatischen Installation lediglich aus dem Bestätigen des Anlegens der Datenbank besteht, durchgeführt und abgeschlossen werden. Ab diesem Punkt ist der normale Einsatz von Apache Roller möglich.

9.2.2. Einbau der Storage-API

Die Anbindung der Storage API wurde analog zu der Realisierung beim Demonstrationsprojekt Content-Battle umgesetzt. Das JAR-Archiv wurde per Maven-Konfiguration eingebunden. Aus Zeitgründen wurde die Einbindung der Storage-API nur prototypisch für den einfachen Upload von Mediendateien für eingerichtete Blogs umgesetzt, statt sie an allen nötigen Stellen einzubauen, an denen auf Dateien zugegriffen wird.

In der Klasse
`org.apache.roller.weblogger.business.FileContentManagerImpl` konnte durch Anpassung der Methode `saveFileContent` für den Upload die prinzipielle Möglichkeit der Einbindung der Storage-API gezeigt werden.

9.2.3. Fazit

Die Schwierigkeit war, bedingt durch die Einarbeitungszeit als Einzelperson in ein fremdes Softwareprojekt, dass die verbleibende Zeit für eine Anbindung der Storage-API relativ knapp war. Somit konnte die API nicht in ihrem vollen Funktionsumfang möglichst homogen in Apache Roller eingebunden werden. Von einem Produktiveinsatz des modifizierten Apache Roller ist in diesem Fall abzuraten. Dieser Sachverhalt stellt jedoch keine Einschränkung der prinzipiellen Einsatzfähigkeit der Storage-API dar.

9.3. jforum

jforum ist die dritte Anwendung, die für Integration der Storage-API ausgewählt wurde. Es handelt sich dabei um ein populäres Forum-Projekt, das in Java implementiert wurde. Es wird unter anderem von Electronic Arts in diversen Spiele-Foren genutzt [21]. Eines dieser Spiele ist „Spore“ [34]. Außerdem benutzt es die bekannte Webseite für Java-Entickler JavaRanch [17, 8]. Es ist optisch an phpBB [27] angelehnt und bietet alle gängigen Funktionen einer Forensoftware [18]. Eine vollständige Liste dieser Funktionen kann auf der Website eingesehen werden.

Eine solche Funktion ist die Möglichkeit, Beiträgen Dateien anzuhängen. An dieser Stelle sollte die Storage-API ansetzen. Anstatt Dateien auf den Webserver zu laden, wie dies in jforum üblich ist (durch die Klasse `DiskFileItem`), sollten die Dateien automatisch auf einen Cloud-Storage geladen werden. Dabei sollte die Ordnerhierarchie erhalten bleiben, ebenso wie etwaige Metadaten zu den Dateien. Die

eine Funktionalität, die angepasst werden musste, war demnach der `Upload`. Auf der anderen Seite müssen die Dateien anschließend wieder gefunden werden. Daher musste auch das Verhalten dafür entsprechend angepasst werden. Dies erwies sich als die größere Herausforderung, wie im weiteren Verlauf verdeutlicht wird.

9.3.1. Download, Deployment und Installation

Bevor die Änderungen im Code des Projekts dargelegt werden, muss zuvor auf das Deployment und die Installation von jforum eingegangen werden. Zunächst lädt man den Source Code von der Hersteller-Seite [19]. Dieser lässt sich anschließend in Eclipse importieren und über das Antfile `build.xml` kompilieren. Dadurch erhält man ein WAR-Archiv, welches man auf einen Tomcat-Server laden kann. Außerdem muss eine Datenbank angelegt werden.

Allerdings machte die anschließende Installation von jforum auf dem Webserver Probleme. Es erschien zunächst eine Eingabemaske, in der man u.a. die Datenbank-Verbindung samt Zugangsdaten eingeben musste. Danach hätte die Installation ordnungsgemäß funktionieren müssen. Es erschien stattdessen jedoch die Fehlermeldung:

```
You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax
to use near 'TYPE=InnoDB' at line 1
```

Eine längere Recherche ergab, dass dieses Problem mit neueren MySQL-Versionen ab der Version 5.1 auftritt (siehe [20]). Abhilfe schaffte schließlich die Maßnahme, dass alle Vorkommen des Befehls `TYPE=InnoDB` in der Datei `config/database/mysql/mysql_db_struct.sql` im Programmordner auf dem Server in den Befehl `ENGINE=InnoDB` abgeändert wurden. Danach verlief die Installation problemlos.

9.3.2. Einbau der Storage-API

Da jforum nicht auf Maven zurückgreift, musste die Storage-API als normales Java-Paket in das Projekt importiert werden. Die für den Datei-Upload zuständigen Klassen befinden sich in den Paketen `net.jforum.util.legacy.commons.fileupload` und `net.jforum.util.legacy.commons.fileupload.disk`. Hier ist insbesondere das Interface `FileItem` hervorzuheben. Dieses verfügt über die zu implementierenden Methoden `write(File)` und `delete()`. Das Interface wird von den Klassen `DefaultFileItem` und `DiskFileItem` implementiert.

Die ursprüngliche Idee war, eine weitere Klasse zu schreiben, die `FileItem` implementiert. An höherer Stelle im Quellcode hätte dann der Aufruf entsprechend abgeändert werden müssen, so dass die alternative Klasse aufgerufen würde. Idealerweise hätte es eine Konfigurationsdatei gegeben, in der die Implementierung angegeben wäre. Bei Broadleaf Commerce war dies noch der Fall, bei jforum allerdings nicht.

Dazu war die Abfolge der Methodenaufrufe im Falle eines Upload undurchsichtig. Überall im Code hätten die Aufrufe umgeschrieben werden müssen.

Daher war es einfacher, die Methoden in der Klasse `DiskFileItem` zu überschreiben. Diese Klasse ist explizit dafür zuständig Input-Streams auf den Webserver-Speicher zu schreiben. Stattdessen sollten nun Aufrufe von entsprechenden Methoden der Storage-API dafür sorgen, dass die Dateien auf den Cloud-Speicher geladen werden.

Erste Tests verliefen erfolgreich. Die Dateien, die über das Forum hochgeladen wurden, kamen auf dem Cloud-Server (getestet mit Google Drive) an. Das Problem lag aber in der Verwaltung der Dateien. Die Pfad-Angaben, die in jforum über `getStoreLocation` abgerufen werden, wurden der Einfachheit halber ignoriert und alle Dateien ohne weitere Ordner-Hierarchie auf den Cloud-Storage geladen. Als unlösbares Problem, zumindest bei vertretbarem Aufwand, erwies sich jedoch die Aufgabe, die Dateien zu einem späteren Zeitpunkt in jforum wieder abrufen zu können. So erschienen `NullPointerExceptions`, sobald eine Datei abgerufen werden sollte. Letztlich erwies sich jforum doch als zu komplex, um dieses Problem im gegeben Zeitrahmen zu lösen. Es war nicht ersichtlich, an welcher Stelle und über welche Parameter Dateien in jforum wieder gefunden und geladen werden. An dieser Stelle müsste man ansetzen und entsprechende Methoden der Storage-API aufrufen.

9.3.3. Fazit

Grundsätzlich ist es möglich, die Storage-API in jforum einzubauen und zu nutzen. Allerdings ist dies mit erheblichen Änderungen im Quellcode verbunden. Zur einfachen Demonstration eines Uploads wurde hier lediglich die Methode `write(File)` ersetzt, allerdings sind viel umfangreichere Bearbeitungen erforderlich, damit die Dateien in jforum korrekt verwaltet, d.h. angezeigt, gelöscht und deren Metadaten gesetzt werden können.

10. Fazit

Cloud-Computing bietet die Möglichkeit, Ressourcen - egal ob Hardware (IaaS), Plattformen (PaaS) oder sogar fertige Anwendungen (SaaS) - nach Bedarf vergleichsweise einfach einzukaufen. Damit ermöglicht es Cloud-Computing, kostengünstige, skalierbare und ausfallsichere Dienste im Web zur Verfügung zu stellen. Diese Vorteile sind der Grund für den schnellen und großen Erfolg dieser Technologie. Nachteile wie Sicherheit, Compliance-Anforderungen und der Vendor-Lock-In-Effekt werden in der Regel ignoriert bzw. durch den wirtschaftlichen Vorteil in den Hintergrund gedrängt.

10.1. Zusammenfassung

Das Ziel dieser PG war, sich mit dem Vendor-Lock-In-Effekt näher auseinanderzusetzen und mögliche Lösungen auszuarbeiten. Dabei sollten Angebote einzelner Cloud-Anbieter nicht nur innerhalb dieses Anbieters, sondern über Anbietergrenzen hinweg ausgetauscht werden können. Abbildung 7 aus dem Informationsheft zu den Projektgruppen zeigt das Koordinatensystem, in dem wir uns idealerweise frei bewegen sollten. Da dies aber zu komplex für eine einzelne PG war und wir nicht alle Übergänge für sinnvoll hielten, passten wir das Ziel an die Anforderungen des Demonstrationsprojekts an. Um den Vendor-Lock-In-Effekt zu mildern, wurde schrittweise vorgegangen. Dazu wurden zunächst in der Seminarphase überwiegend PaaS-Anbieter untersucht und einige, die sowohl ähnliche Charakteristika aufwiesen als auch unseren Anforderungen genügten, ausgewählt. Zu den Anforderungen an die PaaS-Anbieter gehörten unter anderem die Unterstützung der Programmiersprache Java und die Verwendung eines Webservers, in dem WAR-Archive deployt werden können.

Zusätzlich zu PaaS untersuchten wir auch IaaS näher, schränkten uns dabei aber für die Verwendung auf Storage-Anbieter ein. Hierfür entwickelten wir die Storage-API, eine einheitliche Schnittstelle der von uns ausgewählten Storage-Anbieter. Sie erlaubt es, einen beliebigen Cloud-Storage-Anbieter hinter einem einheitlichen Interface zu verstecken und unifiziert somit die Kommunikation mit den unterschiedlichen Diensten. Zum direkten Test der Storage-API wurde das in Kapitel 7 beschriebene Demonstrationsprojekt Content-Battle erstellt und verwendet.

Beim Content-Battle handelt es sich um eine Webanwendung, bei der registrierte Benutzer ihre hochgeladenen Inhalte (Bilder, Videos und Text) gegeneinander antreten lassen und anderen Nutzern eine Abstimmung darüber ermöglichen.

Einen weiteren Arbeitsschritt stellt das Installer-Projekt dar, mit dem eine einheitliche Schnittstelle zu verschiedenen PaaS-Anbietern realisiert wurde. Der Installer ist als Webapplikation umgesetzt, die sowohl die automatische Konfiguration als auch das automatische Deployen bei unterschiedlichen PaaS-Dienstleistern übernimmt. Die Auswahl der Dienste beschränkt sich auf Anbieter mit möglichst heterogener Serverstruktur und eingesetzter Technologie, um eine breite Anwendbarkeit aufzuzeigen. Die Anbindung weiterer Anbieter ist möglich. Als Machbarkeitsnachweis

wurden abschließend verschiedene quelloffene Projekte bearbeitet, um die Storage-API einzubauen.

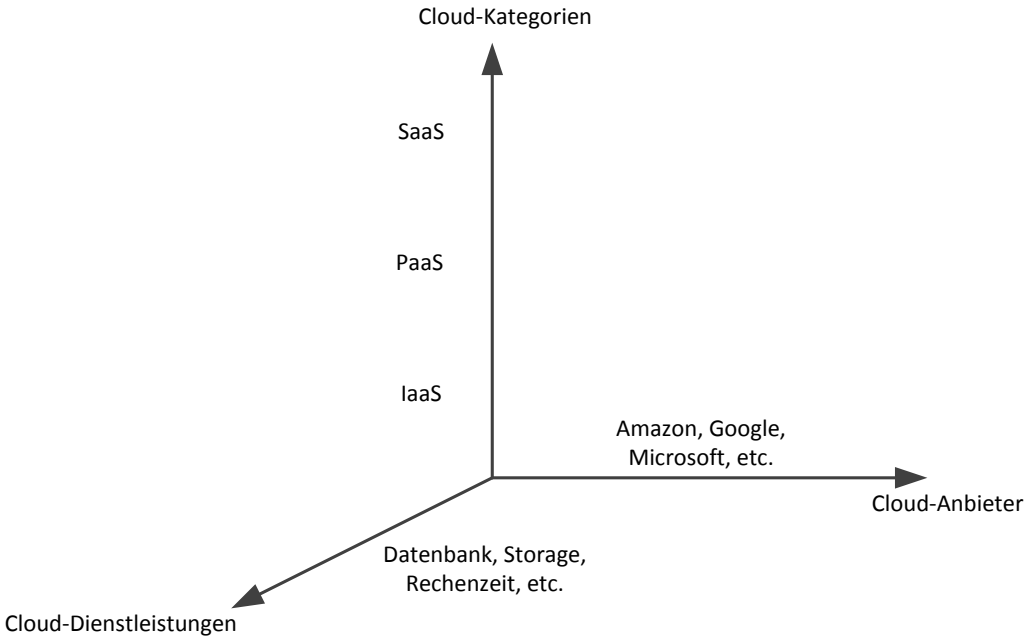

Abbildung 7: Drei wählbare Dimensionen beim Cloud-Computing

10.2. Fazit und Ausblick

Auch wenn diese PG einiges für das gesetzte Ziel ausarbeiten konnte, bleiben Fragen offen. Zwar konnte der Vendor-Lock-In-Effekt in ersten Zügen abgeschwächt werden, jedoch hat sich die Arbeit der PG in Anlehnung an die Abbildung 7 größtenteils auf der PaaS-Ebene und zu einem kleinen Teil, mit der Storage-API, auf der IaaS-Ebene bewegt. Somit wäre ein möglicher nächster Schritt die Betrachtung weiterer Ebenen. Dazu gehört auch eine detailliertere Analyse von IaaS-Anbietern. IaaS-Anbieter stellen dem Kunden nicht nur Storage zur Verfügung, sondern meist vollständige virtuelle Maschinen. Die Untersuchung solcher Anbieter wäre damit durchaus lohnenswert, da der Vendor-Lock-In-Effekt hier noch schwerer zum Tragen kommt. Dies hängt unter anderem damit zusammen, dass sich die Übertragung kompletter virtueller Maschinen zwischen IaaS-Anbietern als sehr problematisch erweist.

Eine weitere Möglichkeit das Thema dieser PG fortzusetzen wäre auch die im Rahmen der PG aufgekommenen Ideen der Datenmigration und des Balancing aufzugreifen und durch andere PGs oder Abschlussarbeiten zu realisieren.

Bei Migration geht es z.B. darum, bei einem Anbieterwechsel die Daten automatisch vom alten zum neuen Anbieter zu transferieren. Das Ganze ist unter anderem

auch für Storage-Anbieter denkbar. Für die Realisierung ist es sinnvoll an der hier vorgestellten Storage-API anzuknüpfen.

Das Balancing im Rahmen von Storage lässt sich unter folgendem Szenario motivieren. Der Kunde *Spar Sam* bietet eine Webseite an, bei der jeder ein Fotoalbum erstellen und anschliessend seine Bilder hochladen kann. Spar Sam ist sich der Vorteile von Cloud-Computing bewusst und nutzt sogar unsere Storage-API. Kosteneffizient orientiert wünscht sich Spar Sam, dass neu erstellte Alben immer den derzeit günstigsten Storage-Anbieter nutzen. Genau hier kommt das Balancing, in diesem Fall als Pricebalancing, ins Spiel und verteilt die Daten nach Spar Sams Wünschen optimal. Natürlich wären auch andere Arten des Balancing denkbar. Man könnte das Balancing auch mit der Migration verbinden, um die kompletten Daten immer bei dem derzeit günstigsten Anbieter abzulegen.

Weiterhin ist es sehr interessant, sich in Bezug auf Intercloud große Cloud-Anbieter anzuschauen. Damit sind insbesondere die marktdominierenden Anbieter wie Amazon und Microsoft gemeint. Hierzu müssten aber die Probleme, die zumindest zum Teil bereits von dieser PG aufgedeckt wurden, behoben werden. Dazu zählen sowohl die finanziellen als auch die rechtlichen Probleme, die mit der Nutzung dieser Anbieter innerhalb einer PG auftreten. Diese Schwierigkeiten ließen sich beispielsweise umgehen, wenn für Veranstaltungen oder Abschlussarbeiten kostenpflichtige Zugänge gestellt würden.

10.3. Erfahrungen & Erkenntnisse

Dieses Unterkapitel befasst sich mit den Erfahrungen und Erkenntnissen, die wir in dieser Projektgruppe im Laufe der zwei Semester machen konnten. Projektarbeit fordert im Allgemeinen eine andere Herangehensweise und zusätzlich einen speziellen Mehraufwand wie die Organisation der Gruppenarbeit. Im Folgenden erläutern wir, was uns im Laufe der PG-Phase an Positivem und Negativem aufgefallen ist und wie wir etwaige Probleme bewältigten.

10.3.1. Was lief schlecht?

Bei Projektarbeiten spielen Projektmanagementwerkzeuge wie Redmine wegen ihrer Funktionalitäten eine sehr bedeutende Rolle. Der richtige Einsatz dieser Werkzeuge bringt erhebliche Vorteile mit sich. Leider hat sich ein Großteil der PG nur auf wenige Funktionen des Redmines, wie das Wiki, beschränkt und z.B. das Ticketsystem nicht ausreichend genutzt. Dabei bietet das Ticketsystem nicht nur den Vorteil Aufgaben zu verteilen und zu dokumentieren, sondern auch im Hinblick auf den Zwischen- und Endbericht eine gute Nachvollziehbarkeit der einzelnen Vorgänge insbesondere der Milestones.

Ein weiteres Problem, das uns als Gruppe auffiel, ist die langsame Kommunikation zwischen einzelnen Teilgruppen wie z.B. zwischen Frontend- und Backend-Entwicklern. Zwar bietet Redmine einige Möglichkeit der Kommunikation unter den Teilnehmern, wie das Ticketsystem, allerdings sind diese Möglichkeiten nur langsam

bzw. unzureichend. Die Email als eins der einfachsten Kommunikationsmittel, die allen zur Verfügung steht, konnte das Problem auch nicht lösen, da das Abrufen der Emails von Teilnehmer zu Teilnehmer in unterschiedlichen Zeitintervallen gehandthabt wurde und sie sich nicht für alle Problembeschreibungen eignen. Dieses Problem hat uns vor allem im ersten Semester viel Zeit gekostet, da viele, auch eher kleinere Probleme, erst in der PG-Sitzung, also von Woche zu Woche, angesprochen und sofern möglich auch sofort behoben wurden. Im zweiten Semester fanden wir für dieses Problem eine gute Lösung, die im nächsten Abschnitt 10.3.2 beschrieben wird.

Das Verfehlen von Deadlines hat sich leider auch in dieser PG eingeschlichen und führte meist zu Terminverschiebungen. Hier kann auch Redmine, trotz der guten Möglichkeiten Milestones einzutragen, nicht wirklich helfen, da es sich in der Regel um ein Einstellungsproblem des jeweiligen Teilnehmers handelt. Allerdings sollte man auch in Betracht ziehen, dass die Zeitplanung möglicherweise nicht alle Gegebenheiten berücksichtigte.

Ein spezielles Problem dieser PG war der mehrfache Wechsel der Betreuer. Er machte sich bei der Erstellung des Zwischen- und Endberichts stark bemerkbar, da sich z.B. die Korrekturzeit in die Länge zog und uns damit zusätzlich die Einhaltung der Deadlines erschwerte.

Ein allgemein wichtiger Punkt bei Softwareprojekten ist das Testen, dass von uns leider zu wenig Aufmerksamkeit bekam. So führten z.B. fehlende Tests zu Unsicherheit bezüglich der Korrektheit von Backend-Services bei der Verwendung im Frontend von Content-Battle. In einigen Fällen wurden Fehler im Frontend gesucht, obwohl der Fehler in den Backend-Services zu finden war und umgekehrt. Diese Problematik trieb uns schließlich dazu, Tests für die Backend-Services zu implementieren, damit das Frontend-Team seine Arbeit weiterführen und schließlich auch beenden konnte.

10.3.2. Was lief gut?

Der möglicherweise wichtigste Punkt bei Projektarbeiten, insbesondere der Gruppenarbeit, ist die Kommunikation. Wie bereits im Abschnitt 10.3.1 beschrieben, gab es in dieser PG eingige Probleme diesbezüglich. Als Lösungsvorschläge erwiesen sich folgende Punkte

- Sondertreffen (zusätzlich zur wöchentlichen PG-Sitzung)
- Tägliche Treffen (der betroffenen Teilnehmer, in der Regel der Teilgruppe z.B. des Frontend-Teams, falls nötig auch der ganzen Gruppe)
- Einzeltreffen zweier PG-Mitglieder (für spezielle Problemlösungen)

als zielführend. Da aus unterschiedlichen, meist zeitlichen, Gründen das Treffen vor Ort nicht klappte, entschieden wir uns dafür, dieses Problem durch Videokonferenzen zu lösen. In Google Hangout Sitzungen konnten Probleme schnell und dezentral gelöst werden. Dies setzte sich als gutes Konzept durch. Wann immer solche Sitzungen

nicht ausreichend waren, trafen sich zwei der beteffenden PG-Mitglieder miteinander oder ein Sondertreffen zum gemeinsamen Programmieren der betroffenen Teilnehmer wurde vereinbart. Egal welches Vorgehen gewählt wird, die PG-Gruppenarbeit erfordert ein hohes Maß an Disziplin und Termintreue, die wir uns erst im Laufe der Projektgruppe angeeignet haben.

A. Anhang

A.1. Vergleichsmatrix

Cloudanbieter	Entwicklung	Deployment	Einschränkungen
Windows Azure	Java mit Eclipse (-Plugin) + SDK; Tomcat; Tutorials	WAR-Datei, Eclipse-Plugin, Kommandozeile nur unter Windows	
HP			Object Storage (SQL und Block Storage in private Beta)
Amazon	Java mit Eclipse und dafür bereitgestelltem SDK; Viele Tutorials; Tomcat Server	WAR Datei per Web-Interface, Eclipse Plugin oder Kommandozeile hochladen	keine
Heroku	Java mit Eclipse; Andere IDEs möglich; Tutorials, etc. auf Eclipse ausgelegt; Java SE, Servlets, Services möglich	Deployment via Git; Alle Funktionen von Git können benutzt werden (Branching, etc.)	
Cloud Bees	Java	Eclipse-Plugin, SDK, WAR-Datei	max. 5 apps
OpenShift	Java (JBoss Application Server 7.1), PHP, Perl; Videoblogs, Tutorials	Git, SSH-Direktzugriff	max 3. Gears
Google App-Engine	Entwicklung von Java Webanwendungen mit Eclipse-Plugin, SDK; Andere IDEs möglich; Tutorials und Demos auf Eclipse ausgelegt; Servlets, JSP, JDO, ...	Eclipse-Plugin, Kommandozeile	Schreiben in Dateisystem, Sockets, Zeitlimit, Threads, ...
Cloud Foundry	Java Entwicklung mit Support für Spring Framework(andere Frameworks nicht offiziell unterstützt, funktionieren aber teilweise)	WAR File push via CL-Tool oder Eclipse Plugin	beta (kostenlos, noch kein Preismodell für Produktiveinsatz vorgestellt)

Tabelle 5: Vergleichsmatrix

A.2. Testplan

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	Anmeldeinformationen werden im Konstruktor korrekt gesetzt / erzeugt	Ungültiger Login	GoogleDrive: Dominik; Miguel
2	Positivtest	Logindaten inkorrekt - Exception	Fehler	GoogleDrive: Dominik; Miguel
3	Negativtest	Keine Verbindung vorhanden - Aussagekräftige Exception	Fehler	Miguel
4	Negativtest	Dienst nicht verfügbar - Aussagekräftige Exception	Fehler	Miguel

Tabelle 6: Allgemeine Tests

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	Bytes werden korrekt hochgeladen	Ende des Streams evtl. fehlerhaft	Dominik M.
2	Positivtest	Metadaten werden korrekt übernommen	Metadaten werden nicht korrekt übernommen	Dominik M.
3	Negativtest	Leere Dateien können nicht hochgeladen werden	Upload ist eventuell doch möglich	Dominik M.
4	Negativtest	Spezialfall Metadaten: MimeType	MimeType wird nicht gespeichert	Dominik M.
5	Negativtest	Speicherkontingent wird korrekt erkannt	Speicherkontingent wird nicht korrekt erkannt	Dominik M.
6	Negativtest	Ungültige parentId - Exception oder null als Rückgabe	Fehler	Dominik M.

Tabelle 7: String uploadFile(InputStream in, IMetaData meta, String parentId)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	Bytes werden korrekt runtergeladen	Ende des Streams evtl. fehlerhaft	Fabian
2	Negativtest	Falsche fileId wird übergeben - Exception oder null als Rückgabe	Fehler	Fabian
3	Negativtest	Metadaten werden korrekt empfangen	Metadaten werden nicht korrekt empfangen	Fabian
4	Negativtest	Downloadkontigent wird korrekt gesetzt (Mediafire)	Downloadkontigent wird nicht korrekt gesetzt	

Tabelle 8: OutputStream downloadFile(String fileId)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	Metadaten werden korrekt übermittelt	Fehlerhafte Metadaten	Fabian
2	Negativtest	Falsche fileId - Exception oder null als Rückgabe	Fehler	Fabian

Tabelle 9: IMetaData getFileHeader(String fileId)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	File wird korrekt verschoben	File wird kopiert oder nicht verschoben	Dominik S.
2	Negativtest	Falsche fileId - Exception	Fehler	Dominik S.
3	Negativtest	Falsche parentDestination - Exception	Fehler	Dominik S.

Tabelle 10: void moveFile(String fileId, String parentDestination)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	File wird gelöscht	File wird nicht gelöscht	Dominik M.
2	Negativtest	Falsche fileId - Exception	Fehler	Dominik M.

Tabelle 11: void deleteFile(String fileId)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	File wird korrekt modifiziert	File wird nicht modifiziert	Sebastian F.
2	Positivtest	Ungültige Metadaten - Exception	Fehler	Sebastian F.
3	Negativtest	Metadaten werden nicht korrekt übertragen	Nicht alle Metadaten werden geändert	Sebastian F.
4	Negativtest	Falsche fileId - Exception	Fehler	Sebastian F.

Tabelle 12: void modifyFile(String fileId, IMetaData meta)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	Folder wird erstellt	Metadaten (Name) werden nicht korrekt gesetzt	Niklas
2	Positivtest	Folder wird erstellt	Folder wird nicht erstellt	Niklas
3	Negativtest	Metadaten werden nicht korrekt übertragen	Nicht alle Metadaten werden geändert	Niklas
4	Negativtest	Falsche parentId- Exception oder false als Rückgabe	Fehler	Niklas

Tabelle 13: String createFolder(IMetaData meta, String parentId)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	Ordner wird korrekt gelöscht	Ordner wird nicht gelöscht	Niklas
2	Positivtest	Ordner wird korrekt gelöscht	Ordner wird nicht gelöscht - Methode bricht ab	Niklas
3	Negativtest	Falsche folderId - Exception	Fehler	Niklas

Tabelle 14: void deleteFolder(String folderId)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	Alle Dateien im Ordner werden korrekt angegeben	Nicht alle oder nur Files/nur Folder werden angezeigt	Tilo
2	Positivtest	Folder ist leer - null als Rückgabe	Fehler	Tilo
3	Negativtest	Dateien in Unterordnern werden mit angezeigt	Dateien in Unterordnern werden mit angezeigt	Tilo
4	Negativtest	Falsche folderId - Exception oder null als Rückgabe	Fehler	Tilo

Tabelle 15: List<IMetaData> listFolder(String folderId)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	Folder wird korrekt modifiziert	Folder wird nicht korrekt modifiziert	Jaouad
2	Positivtest	Ungültige Metadaten - Exception oder false als Rückgabe	Fehler	Jaouad
3	Negativtest	Metadaten werden nicht korrekt übertragen	Nicht alle Metadaten werden geändert	Jaouad
4	Negativtest	Falsche folderId - Exception oder null als Rückgabe	Fehler	Jaouad

Tabelle 16: void updateFolder(String folderId, IMetaData meta)

Test	Testart	Erwartet	Mögliche Probleme	Tester
1	Positivtest	StorageInfo wird korrekt zurückgegeben	StorageInfo kann nicht angefragt werden	David
2	Positivtest	Einzelne StorageInfo Felder sind vorhanden	Felder nicht gesetzt	David

Tabelle 17: IStorageInfo getStorageInfo()

A.3. Backend Webservices

URL	Funktion	HTTP Verb
Battle		
/service/battle/id	Gibt ein BattleDTO in JSON zurück.	GET
/service/battle	Anhand von JSON oder XML wird ein Battle aktualisiert. Gibt das aktualisierte BattleDTO in JSON zurück.	PUT
/service/battle	Anhand von JSON oder XML wird ein Battle erstellt und gibt das erstellte BattleDTO in JSON zurück.	POST
/service/battle/id	Löscht ein Battle.	DELETE
/service/battle/running	Gibt eine JSON Liste von laufenden BattleDTOs zurück.	GET
Content		
/service/content/id	Gibt ein ContentDTO in JSON zurück.	GET
/service/content	Anhand von JSON oder XML wird ein Content aktualisiert. Gibt das aktualisierte ContentDTO in JSON zurück.	PUT
/service/content	Anhand von JSON oder XML wird ein Content erstellt und gibt das erstellte ContentDTO in JSON zurück.	POST
/service/content/id	Löscht einen Content.	DELETE
/service/content/user/id	Gibt eine JSON Liste aller Contents eines Users zurück.	GET
/service/content/battle/id	Gibt eine JSON Liste aller Contents eines Battles zurück.	GET
Matchup		
/service/matchup/id	Gibt ein MatchupDTO in JSON zurück.	GET
/service/matchup/battle/id	Gibt eine JSON Liste aller Matchups eines Battles zurück.	GET
/service/matchup/content/id	Gibt eine JSON Liste aller Matchups zu einem bestimmten Content zurück.	GET
/service/matchup/user/id	Gibt eine JSON Liste aller Matchups in denen ein User abgestimmt hat zurück.	GET

Tabelle 18: Services für das Backend (Battle, Content, Matchup)

URL	Funktion	HTTP Verb
KOMatchup		
/service/komatchup/id	Gibt ein KOMatchupDTO in JSON zurück.	GET
/service/komatchup/random/battleId	Gibt ein zufälliges KOMatchupDTO zu einem Battle in JSON zurück.	GET
/service/vote/userId/matchupId/contentId	Ermöglicht einem User auf einen Content in einem Matchup zu voten. Gibt das geänderte KOMatchupDTO zur matchupId nach dem Vote in JSON zurück.	GET
User		
/service/user/id	Gibt ein UserDTO in JSON zurück.	GET
/service/user	Anhand von JSON oder XML wird ein User aktualisiert. Gibt das aktualisierte UserDTO in JSON zurück.	PUT
/service/user	Anhand von JSON oder XML wird ein User erstellt und gibt das erstellte UserDTO in JSON zurück.	POST
/service/user/id	Löscht einen User.	DELETE
RegUser		
/service/reguser/id	Gibt ein RegUserDTO in JSON zurück.	GET
/service/reguser/name/name	Gibt ein RegUserDTO anhand eines (unique) Usernamen in JSON zurück.	GET
/service/reguser	Anhand von JSON oder XML wird ein RegUser aktualisiert. Gibt das aktualisierte RegUserDTO in JSON zurück.	PUT
/service/reguser	Anhand von JSON oder XML wird ein RegUser erstellt und gibt das erstellte RegUserDTO in JSON zurück.	POST
/service/reguser/id	Löscht einen RegUser.	DELETE
Maintenance		
/maintenance/usercleanup/start	Startet den Scheduler um Gäste zu löschen	POST
/maintenance/usercleanup/stop/delay	Stoppt den Scheduler um Gäste zu löschen nach delay Ausführungen	POST
/maintenance/battle/maintance	Startet neue, verfügbare Battle Phasen	POST

Tabelle 19: Services für das Backend (KOMatchup, User, RegUser, Maintenance)

URL	Funktion	HTTP Verb
Konfiguration		
/maintenance/config/key	Liefert die zu key gespeicherte Konfiguration	GET
/maintenance/config/key	Fügt zu key die im Body übergebene value hinzu	POST
/maintenance/config/key	Ändert die zu key gespeicherte Konfiguration in die im Body übergebene value	PUT
/maintenance/config/key	Löscht die zu key gespeicherte Konfiguration	DELETE
/maintenance/config/all	Liefert alle gespeicherten Konfigurationen als Key-Value Paare	GET
Autorisation		
/j_spring_security_check	Ermöglicht einen Benutzer-Login via Spring Security. Die URL muss als action im Login Formular benutzt werden. Die Eingabefelder müssen für den Namen j_username und für das Passwort j_password heißen. Passwörter werden mit SHA (im Moment ohne Salt) verschlüsselt. Es gibt einen Administratoren Account mit den Credentials Name: Admin Passwort: pg570	POST
/logout	Invalidiert die Session und loggt den aktuellen RegUser defacto aus.	POST
Upload		
/upload/form	Lädt eine Datei hoch und gibt IMetaData zurück. Benötigt enctype="multipart/form-data" und input type="file" name="file"	POST
Download		
/download/fileid	Lädt eine Datei runter.	GET

Tabelle 20: Services für das Backend (Konfiguration, Autorisation, Upload, Download)

Literatur

[1] 4Shared API, . URL http://code.google.com/p/4shared-api/. [Online; Stand 12. August 2013].

[2] 4Shared Java SOAP, . URL http://www.4shared.com/developer/docs/samples/#Java. [Online; Stand 12. August 2013].

[3] HTML enhanced for web apps! URL http://angularjs.org/. [Online; Stand 12. August 2013].

[4] Apache License, Version 2.0, . URL http://www.apache.org/licenses/LICENSE-2.0. [Online; Stand 12. August 2013].

[5] Apache Roller, . URL http://roller.apache.org. [Online; Stand 12. August 2013].

[6] BACKBONE.js. URL http://backbonejs.org. [Online; Stand 12. August 2013].

[7] Open Source eCommerce Framework - Broadleaf Commerce Community. URL http://www.broadleafcommerce.org/. [Online; Stand 12. August 2013].

[8] Java Forums at the Big Moose Saloon. URL http://www.coderanch.com/forums. [Online; Stand 12. August 2013].

[9] Apache Commons Exec. URL http://commons.apache.org/proper/commons-exec/. [Online; Stand 12. August 2013].

[10] Dropbox Core SDK. URL https://www.dropbox.com/developers/core/sdk. [Online; Stand 12. August 2013].

[11] A framework for creating ambitious web applications. URL http://emberjs.com. [Online; Stand 12. August 2013].

[12] Bootstrap. URL http://getbootstrap.com. [Online; Stand 13. September 2013].

[13] Google Drive SDK, . URL https://developers.google.com/drive/downloads. [Online; Stand 12. August 2013].

[14] Integrate your app with Google Drive, . URL https://developers.google.com/drive. [Online; Stand 12. August 2013].

[15] IBM RollerUI. URL https://ibm.com/developerworks/community/blogs/roller-ui. [Online; Stand 12. August 2013].

[16] Inno Setup. URL http://www.jrsoftware.org/isinfo.php. [Online; Stand 12. August 2013].

[17] JavaRanch - a friendly place for Java greenhorns! URL http://www.javaranch.com/. [Online; Stand 12. August 2013].

[18] JForum - Download - heise online. URL http://www.heise.de/download/jforum-1153691.html. [Online; Stand 12. August 2013].

[19] jforum - Powering Communities, . URL http://jforum.net/download.jsp. [Online; Stand 12. August 2013].

[20] MySQL error, . URL https://github.com/rafaelsteil/jforum2/issues/8. [Online; Stand 12. August 2013].

[21] jforum in production - Powering Communities. URL http://jforum.net/production.jsp. [Online; Stand 12. August 2013].

[22] JTrac. URL http://www.jtrac.info/. [Online; Stand 12. August 2013].

[23] A programmer-oriented testing framework for Java. URL https://github.com/junit-team/junit. [Online; Stand 12. August 2013].

[24] The MIT License (MIT). URL http://opensource.org/licenses/mit-license.php. [Online; Stand 12. August 2013].

[25] OpenStack Compute. URL http://www.openstack.org/software/openstack-compute/. [Online; Stand 12. August 2013].

[26] Welcome to Oracle Blogs. URL http://blogs.oracle.com. [Online; Stand 12. August 2013].

[27] phpBB - Free and Open Source Forum Software. URL https://www.phpbb.com/. [Online; Stand 12. August 2013].

[28] The High Velocity Web Framework For Java and Scala. URL http://www.playframework.com/. [Online; Stand 12. August 2013].

[29] Windows PowerShell. URL http://technet.microsoft.com/de-de/library/bb978526.aspx. [Online; Stand 12. August 2013].

[30] Apache R SVN. URL https://svn.apache.org/repos/asf/roller/trunk/app. [Online; Stand 12. August 2013].

[31] Oracle Mojarra JavaServer Faces. URL http://javaserverfaces.java.net/. [Online; Stand 12. August 2013].

[32] SkyDrive SDK. URL http://msdn.microsoft.com/de-de/library/live/hh826521.aspx. [Online; Stand 12. August 2013].

[33] How do I use the SpiderOak API. URL https://spideroak.com/faq/questions/37/how_do_i_use_the_spideroak_web_api/. [Online; Stand 12. August 2013].

[34] The Sporum - The Official Spore Forum. URL http://forum.spore.com/jforum/forums/list.page. [Online; Stand 12. August 2013].

[35] Spring Framework. URL http://www.springsource.org/. [Online; Stand 12. August 2013].

[36] Swing (Java Foundation Classes). URL http://docs.oracle.com/javase/7/docs/technotes/guides/swing/. [Online; Stand 12. August 2013].

[37] Apache Tapestry 5. URL http://tapestry.apache.org/. [Online; Stand 12. August 2013].

[38] Box4J - Open API for Java, 2010. URL http://code.google.com/p/box4j/. [Online; Stand 12. August 2013].

[39] Apache Roller Weblogger, Juni 2012. URL http://www.apache.org/dist/roller/roller-5/v5.0.1/docs/roller-install-guide.pdf. [Online; Stand 12. August 2013].

[40] Box Developers, 2013. URL http://developers.box.com/docs/. [Online; Stand 12. August 2013].

[41] N. Abdelwahd. Entwicklung eines Systems zur bidirektionalen Abbildung zwischen XMLSchema Instanzen und JavaScript Objekten in Web 2.0 Applikationen, Mai 2010. URL http://www.rn.inf.tu-dresden.de/uploads/Studentische_Arbeiten/Belegarbeit_Abdelwahd_Nabil.pdf. [Online; Stand 10. Juli 2013].

[42] Ant. Apache Ant, 2013. URL http://ant.apache.org/. [Online; Stand 10. Juli 2013].

[43] Apache. Apache Maven, 2013. URL http://maven.apache.org/. [Online; Stand 10. Juli 2013].

[44] BITKOM. Cloud Computing - Was Entscheider wissen müssen. Bundesverband Informationswirtschaft, 2010. URL http://www.bitkom.org/files/documents/BITKOM_Leitfaden_Cloud_Computing-Was_Entscheider_wissen_muessen.pdf. [Online; Stand 10. Juli 2013].

[45] BITKOM. Cloud Computing - Umsatz mit Cloud Computing steigt auf fast 8 Milliarden Euro, 2013. URL http://www.bitkom.org/de/presse/8477_75301.aspx. [Online; Stand 13. August 2013].

[46] F. Coerschulte. Seminararbeit PG 570 - Amazon WebServices, 2012.

[47] Darcs. Darcs - Distributed. Interactive. Simple., 2013. URL http://darcs.net/. [Online; Stand 10. Juli 2013].

[48] S. Fechner. Seminararbeit PG 570 - Google App Engine, 2012.

[49] Forbes. Cloud Computing's Vendor Lock-In Problem: Why the Industry is Taking a Step Backward, 2011. URL http://tinyurl.com/7pq2xr9. [Online; Stand 13. August 2013].

[50] Git. Git –distributed-is-the-new-centralized, 2013. URL http://git-scm.com/. [Online; Stand 10. Juli 2013].

[51] Google Inc. AngularJS Developer Guide, März 2013. URL http://docs.angularjs.org/guide/. [Online; Stand 10. Juli 2013].

[52] Gradle. Gradle - Automation Evolved, 2013. URL http://www.gradle.org/. [Online; Stand 10. Juli 2013].

[53] ITWissen.info. ITWissen - Webcontainer, 2013. URL http://www.itwissen.info/definition/lexikon/Webcontainer.html. [Online; Stand 10. Juli 2013].

[54] Kemmerzell Media. Dropbox mit mehr als 175 Millionen Nutzern, Juli 2013. URL http://www.kemmerzell-media.de/10853/dropbox-mit-mehr-als-175-millionen-nutzern/. [Online; Stand 12. August 2013].

[55] P. Kröner. *HTML5. Webseiten innovativ und zukunftssicher*, volume 2. Open Source Press, 2011.

[56] R. J. Lorimer. Hibernate: Understanding Lazy Fetching, August 2005. URL http://www.javalobby.org/java/forums/t20533.html. [Online; Stand 12. August 2013].

[57] Maven. Apache Maven, 2013. URL http://maven.apache.org/. [Online; Stand 10. Juli 2013].

[58] Mercurial. Mercurial - Work easier, Work faster, 2013. URL http://mercurial.selenic.com/. [Online; Stand 10. Juli 2013].

[59] D. Mohr. Seminararbeit PG 570 - Heroku Cloud Application Platform, 2012.

[60] Oracle. Oracle Java, 2013. URL http://www.java.com/de/. [Online; Stand 10. Juli 2013].

[61] C. Pautasso, O. Zimmermann, and F. Leymann. RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. Refereed Track: Web Engineering - Web Service Deployment, 2008. URL http://wwwconference.org/www2008/papers/pdf/p805-pautassoA.pdf. [Online; Stand 10. Juli 2013].

[62] M. C. Perello. Seminararbeit PG 570 - OpenShift by RedHat, 2012.

[63] Redmine. Redmine Online Präsenz, März 2013. URL http://www.redmine.org. [Online; Stand 10. Juli 2013].

[64] S. Sander. Performanceanalyse von SOAP- und REST- basierten Services in einer Linguistic Resources Umgebung, Oktober 2010. URL http://lips.informatik.uni-leipzig.de/files/Diplomarbeit.pdf. [Online; Stand 10. Juli 2013].

[65] D. Schoen. Seminararbeit PG 570 - Apache Deltacloud, 2012.

[66] T. Schulz. Seminararbeit PG 570 - Google Apps Application APIs, 2012.

[67] R. Seiger. Entwurf eines mehrseitig sicheren Cloud Storage Dienstes. Technische Universität Dresden, November 2010. URL http://www.rn.inf.tu-dresden.de/uploads/Studentische_Arbeiten/Belegarbeit_Seiger_Ronny.pdf. [Online; Stand 10. Juli 2013].

[68] D. Sparer. Seminararbeit PG 570 - Apache Libcloud, 2012.

[69] Subversion. Apache Subversion, 2013. URL http://subversion.apache.org/. [Online; Stand 10. Juli 2013].

[70] The Latex Projekt. LaTeX – A document preparation system, 2013. URL http://www.latex-project.org/. [Online; Stand 10. Juli 2013].

[71] S. Venier. Seminararbeit PG 570 - Force.com, 2012.

[72] Wikipedia. Java Archive — Wikipedia, Die freie Enzyklopädie, 2013. URL http://de.wikipedia.org/w/index.php?title=Java_Archive&oldid=119424121. [Online; Stand 10. Juli 2013].

[73] Wikipedia. Web Archive — Wikipedia, Die freie Enzyklopädie, 2013. URL http://de.wikipedia.org/w/index.php?title=Web_Archive&oldid=117024291. [Online; Stand 10. Juli 2013].

[74] E. Wolff. *Spring 3 Framework für die Java-Entwicklung*, volume 3. dpunkt-Verlag, 2010.

[75] J. Zarouali. Seminararbeit PG 570 - CloudBees, 2012.

[76] N. A. Zbick. Seminararbeit PG 570 - Oracle Cloud, 2012.