

Bachelor Thesis:  
Untersuchung zu Möglichkeiten des  
Dateimanagements unter Verwendung verschiedener  
Cloudanbieter

Bachelorarbeit  
von

Markus Paeschke

03. Juni 2013 – 13. August 2013

Prüfungsvorsitzender: Prof. Dr. Albrecht Fortenbacher  
Betreuer: Prof. Dr. Christin Schmidt  
Prof. Dr. Albrecht Fortenbacher

Ich möchte mich bei Prof. Dr. Christin Schmidt für die Betreuung und Begleitung, während der Erstellung dieser Arbeit, bedanken, genauso wie bei Prof. Dr. Albrecht Fortenbacher, für die Hilfestellungen und nützlichen Ratschläge.

Ein besonderer Dank geht an Nora Mudrack, Daniel Ghioreanu und Andreas Buff, die mir nicht nur bei der Bachelorarbeit behilflich waren, sondern mich mein ganzes Studium über unterstützt haben und mir immer wieder Kraft und Motivation zum weitermachen gegeben haben. Danke dafür.

Weiterhin danke ich Grit Schneider, Angela Mehnert und Toralf Kampe für das Korrekturlesen und die hilfreichen Ratschläge bei der Erstellung dieser Arbeit.

Markus Paeschke  
Trachtenbrodtstr. 32  
10409 Berlin

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Berlin, den 11. Juni 2017

(Unterschrift)

---

Markus Paeschke

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>v</b>
<b>Abbildungsverzeichnis</b>	<b>vi</b>
<b>Tabellenverzeichnis</b>	<b>vii</b>
<b>Quellcodeverzeichnis</b>	<b>viii</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Aufbau der Arbeit . . . . .	2
<b>2 Grundlagen</b>	<b>3</b>
2.1 Clouddienste . . . . .	3
2.1.1 Clouddienste aus organisatorischer Sicht . . . . .	4
2.1.2 Clouddienste aus technischer Sicht . . . . .	6
2.2 RAID Systeme . . . . .	8
2.2.1 RAID-0 . . . . .	8
2.2.2 RAID-1 . . . . .	9
2.2.3 RAID-1+0 . . . . .	9
2.3 Kryptographie . . . . .	11
2.3.1 Zufallszahlen . . . . .	11
2.3.2 Symmetrische Verschlüsselungsverfahren . . . . .	11
2.3.3 Hashfunktionen . . . . .	13
2.4 OAuth . . . . .	16
2.4.1 Historischer Hintergrund . . . . .	16
2.4.2 OAuth 1.0 . . . . .	16
2.4.3 OAuth 1.0A . . . . .	19
2.4.4 OAuth 2.0 . . . . .	20
<b>3 Anforderungsanalyse</b>	<b>22</b>
3.1 Systemanforderung . . . . .	22
3.2 Cloudanservices . . . . .	23
3.3 Technologien . . . . .	24
3.4 Use-Case-Analyse . . . . .	25
3.5 Abgrenzung zu bestehenden Systemen . . . . .	25

<b>4</b>	<b>Systementwurf</b>	<b>28</b>
4.1	Programmierungsumgebung . . . . .	28
4.2	Evaluation der Cloudservices . . . . .	29
4.2.1	Authentifizierung . . . . .	30
4.2.2	API . . . . .	31
4.2.3	Rechtliche Aspekte . . . . .	31
4.3	Architektur des Systems . . . . .	33
4.3.1	Datenhaltungs-Schicht . . . . .	35
4.3.2	Anwendungslogik-Schicht . . . . .	35
4.3.3	Präsentationsschicht . . . . .	39
<b>5</b>	<b>Implementierung</b>	<b>42</b>
5.1	Projektstruktur . . . . .	42
5.2	Externe Module . . . . .	43
5.3	Selbst entwickelte Klassen und Funktionssammlungen . . . . .	46
5.4	Realisierung der Datenhaltungs-Schicht . . . . .	50
5.5	Realisierung der Anwendungslogik-Schicht . . . . .	51
5.6	Realisierung der Präsentationsschicht . . . . .	54
<b>6</b>	<b>Evaluation und Demonstration</b>	<b>56</b>
6.1	Evaluation des Systems . . . . .	56
6.2	Demonstration des Systems . . . . .	57
<b>7</b>	<b>Zusammenfassung und Ausblick</b>	<b>61</b>
	<b>Literaturverzeichnis</b>	<b>I</b>

## Abkürzungsverzeichnis

<b>AES</b>	Advanced Encryption Standard
<b>AGB</b>	Allgemeine Geschäftsbedingungen
<b>API</b>	Application Programming Interface
<b>App</b>	Application
<b>CRM</b>	Customer-Relationship-Management
<b>CSS</b>	Cascading Style Sheets
<b>DES</b>	Data Encryption Standard
<b>DOM</b>	Document Object Model
<b>E/A</b>	Ein-/Ausgabe
<b>GB</b>	Gigabyte
<b>GHz</b>	Gigahertz
<b>GUI</b>	Graphical User Interface
<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>IaaS</b>	Infrastructure as a Service
<b>JSON</b>	JavaScript Object Notation
<b>KB</b>	Kilobyte
<b>MB</b>	Megabyte
<b>MD5</b>	Message-Digest Algorithm 5
<b>MIME</b>	Multipurpose Internet Mail Extensions
<b>MIT</b>	Massachusetts Institute of Technology
<b>MVC</b>	Model View Controller
<b>NIST</b>	National Institute of Standards and Technology
<b>NSA</b>	National Security Agency
<b>PaaS</b>	Platform as a Service
<b>RAID</b>	Redundant Array of Inexpensive Disks
<b>REST</b>	Representational State Transfer
<b>upm</b>	Umdrehungen pro Minute
<b>SaaS</b>	Software as a Service
<b>SATA</b>	Serial Advanced Technology Attachment
<b>Sass</b>	Syntactically Awesome Stylesheets
<b>SHA</b>	Secure-Hash-Algorithm
<b>SSH</b>	Secure Shell
<b>SSL</b>	Secure Socket Layer-Technologie
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>XML</b>	Extensible Markup Language

# Abbildungsverzeichnis

1	Verteilung der Teilstücke einer Datei unter Verwendung von RAID-0 . . . . .	8
2	Verteilung der Teilstücke einer Datei unter Verwendung von RAID-1 . . . . .	9
3	Verteilung der Teilstücke einer Datei unter Verwendung von RAID-1+0 . . . . .	10
4	Workflow des OAuth 1.0 Authentifizierungsprozesses . . . . .	18
5	Architekturentwurf von CloudGrid . . . . .	33
6	MVC Architektur . . . . .	34
7	Verteilung der Teilstücke einer Datei unter Verwendung von zwei Cloudservices	38
8	Verteilung der Teilstücke einer Datei unter Verwendung von drei Cloudservices	38
9	Verteilung der Teilstücke einer Datei unter Verwendung von vier Cloudservices	38
10	Wireframe der GUI . . . . .	40
11	Schema der SQLite Datenbank . . . . .	51
12	Das Logo von CloudGrid . . . . .	55
13	Einstellungsseite beim ersten Start der Anwendung . . . . .	57
14	Seite zum Verbinden und Bearbeiten der Cloudservices . . . . .	58
15	Die Ordnerüberwachung ist deaktiviert . . . . .	59
16	Die Ordnerüberwachung wurde gestartet . . . . .	59
17	Informationsseite von CloudGrid . . . . .	60

# Tabellenverzeichnis

1	Evaluierte Cloudservices . . . . .	30
---	------------------------------------	----



# Quellcodeverzeichnis

1	Aufbau eines Response vom Service Provider an den Consumer . . . . .	21
2	Ausschnitt des Dropbox Routings . . . . .	44
3	Aufbau einer mit Hogan erstellten HTML Seite . . . . .	45
4	Beispiel eines HTTP Multipart Bodies . . . . .	47
5	JSON String eines Cloudservices . . . . .	50

# Kapitel 1

## Einleitung

### 1.1 Motivation

Clouddienste erfreuen sich in den letzten Jahren einer rasant wachsenden Beliebtheit. Der Umsatz in Deutschland wird, laut einer Prognose von Bitkom, im Jahr 2013 um 47% gegenüber dem Vorjahr ansteigen und liegt damit bei 7,8 Milliarden Euro[vgl. Web13, Seite 2]. Ebenfalls geht aus dieser Prognose hervor, dass sich auch in den kommenden Jahren ein ähnlich starkes Wachstum fortsetzt. So ist es nicht verwunderlich, dass immer mehr Anwendungen einen Upload der Daten in die Cloud ermöglichen oder gar komplett die Datenspeicherung in die Cloud verlagern.

Von einem einfachen Datenbackup wie bei Dropbox<sup>1</sup> oder Google Drive<sup>2</sup>, zum Musikstreaming wie Spotify<sup>3</sup>, bis hin zum kollaborativen Arbeiten in den Google Docs<sup>4</sup>, die Anwendungsgebiete sind dabei vielfältig und wachsen stetig. Google hat mit seinem Chromium OS<sup>5</sup> darüber hinaus ein cloudfähiges Betriebssystem auf den Markt gebracht. Dieses setzt ausschließlich auf Cloudanwendungen und ist ohne einen Internetanschluss nicht nutzbar.

Jedoch sind viele Benutzer skeptisch bei der Verwendung von Clouddiensten. So geht aus einer Studie von Pierre Audoin Consultants hervor, dass 76% der Befragten Bedenken in Bezug auf Sicherheit und Datenschutz bei der Verwendung von Clouddiensten haben [vgl. Pie13, Seite 23]. Bereits im Jahr 2010 gaben 72% der deutschen Teilnehmer in einer Studie von Fujitsu an, dass sie befürchten, dass der Staat ihre Daten in der Cloud einsehen kann[vgl. Fuj10, Seite 05]. Darüber hinaus besteht jederzeit die Gefahr eines Datenverlustes. Die möglichen Szenarien sind dabei vielfältig. Die Anbieter, bei denen die Daten gespeichert werden, können ihren Dienst einstellen, wie es beispielsweise bei Megaupload im Jahr 2012 der Fall war[vgl. The12]. Auch denkbar ist, dass sich Hacker Zugriff auf das System verschaffen und somit an die Daten der Nutzer gelangen.

An diesem Punkt setzt die Bachelorarbeit an. Sie soll aufzeigen, welche Möglichkeiten sich für einen Benutzer bei der Verwendung von verschiedenen Cloudanbietern ergeben und wie in

---

<sup>1</sup><http://www.dropbox.com>

<sup>2</sup><http://drive.google.com>

<sup>3</sup><http://www.spotify.com>

<sup>4</sup><http://docs.google.com>

<sup>5</sup><http://www.chromium.org/chromium-os>

diesem Zusammenhang eine höhere Datensicherheit gewährleistet werden kann. Dazu werden Probleme bestehender Cloudservices aufgezeigt und Lösungskonzepte für selbige erarbeitet.

## 1.2 Zielsetzung

Ziel dieser Arbeit ist es, einen Prototypen, im folgenden CloudGrid genannt, zu entwickeln, der einerseits eine größere Datensicherheit für Dateien in der Cloud gewährleistet, andererseits den Speicherplatz von Cloudanbietern zusammenfasst und diesen dadurch vergrößert. Die Datensicherheit soll durch mehrere Methoden erhöht werden. Zum einen sollen die Daten in den gebündelten Clouds redundant gespeichert werden, was den Vorteil hat, dass ein Service ausfallen und der Benutzer trotzdem über alle Daten verfügen kann. Weiterhin werden Dateien komprimiert, zerteilt und danach auf verschiedenen Cloudanbietern gespeichert. Das hat zur Folge, dass Daten nie vollständig bei einem Anbieter gespeichert werden und dadurch für Dritte nur bedingt brauchbar sind. Zugleich werden diese noch komprimiert, sodass mehr Speicherplatz zur Verfügung steht. Darüber hinaus werden alle Dateien verschlüsselt. Der Inhalt der Daten soll somit geschützt werden.

CloudGrid soll als reine Desktopanwendung umgesetzt werden und damit komplett auf einen Serverdienst verzichten. Dies hat den Vorteil, dass sich der Benutzer nicht bei einem weiteren Service anmelden und seine Daten erneut an eine weitere Institution schicken muss. Dabei soll im Rahmen dieser Arbeit lediglich die Datenverwaltung mit einem Client zugelassen werden. Eine Synchronisierung über mehrere Clients erhöht die Komplexität und wird für eine etwaige spätere Version vorgesehen.

## 1.3 Aufbau der Arbeit

Zu Beginn werden Grundlagen zu Clouddiensten, Sicherheit und Dateihandling aufgezeigt. Anschließend werden die Anforderungen an den zu entwickelnden Prototypen definiert, sowie deren Besonderheiten und Möglichkeiten dargestellt. Darauf basierend wird die Architektur des Systems dargestellt und abschließend die konkrete Umsetzung erläutert. Zudem wird die Funktionsweise des Prototypen demonstriert und auf Defizite hingewiesen. Abschließend wird eine Zusammenfassung über die gewonnenen Kenntnisse und ein Ausblick auf Erweiterungsmöglichkeiten dieser Thematik erörtert.

# Kapitel 2

## Grundlagen

### 2.1 Clouddienste

Das Thema Cloud-Computing ist momentan allgegenwärtig in der Informatik. Dennoch gibt es keine standardisierte Definition, was die „Cloud“ ist. Das Wort „Cloud“ oder im deutschen „Wolke“ weist jedoch auf zwei wichtige Konzepte hin, nämlich Virtualisierung und Skalierbarkeit[vgl. Hö11, Seite 35]. Bei der Virtualisierung werden Computerressourcen transparent zusammengefasst beziehungsweise aufgeteilt. Dadurch wird eine beliebige Sicht auf die Infrastruktur geschaffen, wodurch keine systembedingten Abhängigkeiten für eine Anwendung entstehen[vgl. BKNT10, Seite 2].

Der zweite wichtige Aspekt, die Skalierbarkeit, ermöglicht es, zusätzliche Ressourcen ohne großen Aufwand in das System zu integrieren, sei es um Hardware nachzurüsten oder bestehende Systeme zusammen zu fassen. Das hat den Vorteil, dass ein Unternehmen nicht mehr in diese investieren und selbst verwalten muss. Es kann sich auf die Umsetzung seiner Geschäftsidee konzentrieren und bei wachsenden Anforderungen Ressourcen flexibel vom Provider beziehen[vgl. BKNT10, Seite 2].

Als großer Kritikpunkt lassen sich hingegen die Sicherheitsbedenken aufführen. Dadurch, dass die Anwendungslogik an eine externe Firma ausgelagert wird, verliert das Unternehmen die Kontrolle über seine Daten. Zudem muss es sich auf die Sicherheitsmaßnahmen der Anbieter verlassen und hat keinen Einfluss auf selbige. Besonders bei sensiblen Daten, wie Krankenakten von Patienten oder Kundendaten einer Bank, spielen Sicherheitsaspekte eine übergeordnete Rolle, wodurch unter Umständen von der Verwendung von Cloud-Computing abgeraten werden muss. Auch die Verfügbarkeit und Performance von Ressourcen in einer Cloud geben Grund zur Kritik. Sollte ein Service für mehrere Stunden nicht verfügbar sein, so kann ein erheblicher wirtschaftlicher Schaden für die betroffene Firma entstehen. Zudem existieren momentan keine Standards bei der Implementierung von Ressourcen, womit ein schneller Wechsel des Anbieters in der Regel nicht einfach möglich ist[vgl. BKNT10, Seite 3].

Je nach Anwendungsfall muss ein Unternehmen sich daher detailliert überlegen, ob es Cloud-Computing verwenden möchte und welche Form und welcher Anbieter dabei seinen Ansprüchen genügt.

Neben der Auslagerung von Anwendungslogiken gibt es mehrere Dienste die vorgefertigte

Lösungen anbieten. So streamen Spotify<sup>1</sup> oder Deezer<sup>2</sup> Musik, Youtube<sup>3</sup> streamt Videos, Google Docs<sup>4</sup> bietet eine komplette Office Suite zum kollaborativen Arbeiten an und sogar Bildbearbeitung ist mit Diensten wie dem Photoshop Express Editor<sup>5</sup> möglich. Die Vorteile für den Benutzer liegen dabei auf der Hand. Er kann mobil und über verschiedene Geräte hinweg seine Daten abrufen und bearbeiten, ohne dabei auf zusätzliche Hardware wie USB-Sticks oder DVDs zurückgreifen zu müssen. Darüber hinaus muss sich der Anwender nicht mehr manuell um die Aktualisierung seiner Daten auf verschiedene Geräte kümmern. Beispielsweise bearbeitet er ein Foto in der Cloud, speichert dieses dort und kann es später von einem anderen Rechner herunterladen. Dadurch, dass sich der Anbieter des Clouddienstes um ein sinnvolles Datenbackup kümmern muss, beispielsweise durch redundante Datenhaltung auf mehreren Servern, wird für den Benutzer eine Ausfallsicherheit gewährleistet. Sollte der Rechner defekt sein, gehen die Daten nicht verloren, sondern sind weiterhin auf dem Server verfügbar.

Zudem kann die Architektur von Clouddiensten aus zwei Perspektiven betrachtet werden. Einerseits aus organisatorischer Sicht und andererseits aus technischer Sicht[vgl. BKNT10, Seite 25]. Zu den erstgenannten zählen die „Private Cloud“, die „Public Cloud“ und die „Hybrid Cloud“[vgl. BKNT10, Seite 25]. Clouddienste, die aus technischer Sicht betrachtet werden, können in „Infrastructure as a Service (IaaS)“, „Platform as a Service (PaaS)“ und „Software as a Service (SaaS)“ eingeteilt werden[vgl. BKNT10, Seite 25].

### 2.1.1 Clouddienste aus organisatorischer Sicht

Bei der organisatorischen Betrachtung von Clouddiensten, werden Benutzer und Anbieter strukturell voneinander getrennt. Der Benutzer verwendet dabei einen Dienst, welcher vom Anbieter offeriert wird. Beide Einheiten können dem selben Unternehmen angehören, werden jedoch separat betrachtet.

#### Private Cloud

Unter „Private Cloud“ versteht man eine Cloud, bei der sowohl die Benutzer als auch der Anbieter derselben organisatorischen Einheit angehören[vgl. BKNT10, Seite 25 f.]. Es ist also ein geschlossenes Netzwerk, wie beispielsweise ein firmeninternes Intranet, welches den Benutzern exklusive Ressourcen zur Verfügung stellt[vgl. Hö11, Seite 40]. Hauptgrund für die Verwendung von „Private Clouds“ sind Sicherheitsaspekte. Gerade bei sensiblen Daten ist diese Art der Cloud von großem Vorteil, da sich dadurch viele Sicherheitsprobleme beseitigen lassen. So bleibt die Kontrolle über die Daten bei der Organisation selbst und wird nicht an Dritte weitergegeben[vgl. BKNT10, Seite 26]. Außerdem können einzelne Ressourcen kontrollierter an Benutzer weitergegeben werden. Ein Mitarbeiter erhält so zwar Zugang zu seinem eigenen Dienstplan, jedoch nicht zu dem seiner Kollegen, wohingegen der Geschäftsführer Zugriff auf alle Dienstpläne hat. Auch Hardwareressourcen können flexibel skaliert und entsprechend der aktuellen Situation angepasst werden.

---

<sup>1</sup><http://www.spotify.com>

<sup>2</sup><http://www.deezer.com>

<sup>3</sup><http://www.youtube.de>

<sup>4</sup><https://docs.google.com/>

<sup>5</sup><http://www.photoshop.com/tools/expresseditor>

Jedoch muss die Verwaltung und Wartung der Cloud, sowohl soft- als auch hardwareseitig, von der Organisation selbst übernommen werden. Dieser Aspekt ist aus unternehmerischer Sicht kostenintensiver und daher möglicherweise unattraktiver. Besonders da diese, je nach Umfang, von einem oder mehreren Administratoren gepflegt werden muss.

## Public Cloud

Eine „Public Cloud“ kann als ein am Markt angebotener Clouddienst angesehen werden[vgl. Hö11, Seite 38]. Es sind also Cloud-Angebote, bei denen der Anbieter und die potenziellen Benutzer nicht zu ein- und derselben organisatorischen Einheit gezählt werden[vgl. BKNT10, Seite 25]. Meist sind jene Anbieter gemeint, wenn von der „Cloud“ gesprochen wird. Die Ressourcen solcher Dienste werden meist von vielen Nutzern gleichzeitig geteilt, welche nur durch die Virtualisierung als eigene Umgebung für den einzelnen Benutzer erscheinen. Viele Dienstleister bieten darüber hinaus ein Web-Portal oder gar eine Application Programming Interface (API) zum Interagieren mit dem Dienst an. Diese sind für den Betreiber leicht skalierbar, sodass weitere Hardwareressourcen jederzeit eingebunden werden können. Besonders bei wachsenden Anforderungen an das System ist dieser Aspekt essenziell. Auf Seiten des Nutzers lässt sich dieses Prinzip nur schwer anwenden. „Eine Anpassung an spezifische Anwenderanforderungen ist üblicherweise nur sehr eingeschränkt möglich/erwünscht bzw. steht den Skalierungs- und Effizienzinteressen des Betreibers gegenüber“[Hö11, Seite 39].

Eine übliche Bezahlmethode ist „Pay-per-Use“, das bedeutet je nach benutzten Ressourcen fallen mehr oder weniger hohe Kosten an, sowie die „Flatrate“, für die der Nutzer einen festen Betrag bezahlt, wobei er einen vordefinierten Umfang der Ressourcen zur Verfügung gestellt bekommt. Ein prominenter Vertreter für „Pay-per-User“ ist Amazon EC2<sup>6</sup>, wohingegen beispielsweise Dropbox<sup>7</sup> oder Spotify<sup>8</sup> auf ein Flatrate-Model setzen.

Gerade im Gegensatz zu „Private Clouds“ ergeben sich bei den „Public Clouds“ zahlreiche Bedenken. So muss für jedes System „hinsichtlich Compliance, Sicherheit, Verfügbarkeit, aber auch Performance“[Hö11, Seite 40]. detailliert geprüft werden, ob eine „Public Cloud“ den Anforderungen gerecht wird. Durch den Verlust der, im Vergleich zu den „Private Clouds“, absoluten Kontrolle über die Daten des Benutzers, müssen beim Verstoß gegen vertraglich vereinbarte Regelungen, diese erst juristisch durchgesetzt werden[vgl. Hö11, Seite 40]. Besonders der zeitliche Aspekt von Gerichtsverfahren birgt ein großes Risiko für Unternehmen.

## Hybrid Cloud

„Hybrid Clouds“ sind eine Mischform aus „Public Clouds“ und „Private Clouds“. Hierbei wird eine „Private Cloud“ auf organisationseigenen Ressourcen betrieben, jedoch werden zusätzlich Services herkömmlicher Datenverarbeitungslösungen mittels APIs von „Public Cloud“ Anbietern eingebunden. Somit können konkrete Anforderungen an ein System für das jeweilige Unternehmen angepasst werden. Etwaige Sicherheitsvorgaben können eingehalten werden und gleichzeitig Skalierungseffekte genutzt werden[vgl. Hö11, Seite 42]. Durch Virtualisierung wird auch bei „Hybrid Clouds“ dem Benutzer ein geschlossenes System dargestellt, sodass dieser

---

<sup>6</sup><http://aws.amazon.com/de/ec2>

<sup>7</sup><https://www.dropbox.com/business>

<sup>8</sup><https://www.spotify.com/de>

weder einen Unterschied noch einen Übergang zwischen den einzelnen Clouds bemerkt[vgl. Hö11, Seite 11].

Als Nachteil kann eine erhöhte Komplexität bei der Integration solcher Systeme aufgeführt werden. Eine nahtlose und für den Benutzer transparente Implementierung muss sorgfältig geplant und von den entsprechenden Administratoren und Entwicklern eingebunden werden. Zudem muss detailliert geprüft werden, welche Komponenten in die „Public Clouds“ auszulagern sind, damit die Nachteile selbiger nicht das Gesamtsystem beeinträchtigen.

## 2.1.2 Clouddienste aus technischer Sicht

Bei der technischen Betrachtung der Clouddienste wird eine abstrahierte Sicht auf Hard- und Software geboten. Der Anbieter kümmert sich um die Wartung und Pflege der Ressourcen, wohingegen der Kunde virtualisierte Ressourcen angeboten bekommt.

### Infrastructure as a Service

Einem Benutzer wird bei der Verwendung der IaaS-Schicht eine abstrahierte Sicht auf die Hardware, also unter anderem auf ein bestehendes Netzwerk, einen Server und deren räumliche und klimatische Bedingungen oder auch nur auf Massenspeichermedien, geboten[vgl. Hö11, Seite 47]. Dabei wird dem Kunden meist eine Teilmenge der eigentlichen Ressourcen zur Verfügung gestellt. „Man könnte IaaS auch als eine im Wesentlichen durch Einsatz von Software abstrahierte und virtualisierte Rechenzentrumsressource auffassen“[Hö11, Seite 47]. Das bedeutet, dass sich meist mehrere Kunden ein und dieselbe Ressource teilen, welche lediglich auf Softwareebene virtualisiert wird. Besonders in Hinsicht auf die Skalierbarkeit der Ressourcen ist das ein erheblicher Vorteil.

Als Beispiel für solche Services kann man virtuelle Server von Hosting Providern oder auch Dienste wie Amazon EC2 und Dropbox nennen. Letztere bieten darüber hinaus noch Schnittstellen zur Kommunikation mit den Diensten an, wohin gegen virtuelle Server meist über einen Secure Shell (SSH) Zugang verfügen.

### Platform as a Service

Die PaaS-Schicht baut auf der IaaS-Schicht auf und bietet dem Benutzer Anwendungen, Datenbanken oder Webservices, mit denen er arbeiten kann. Oftmals wird diese auch als »Middleware« bezeichnet[vgl. Hö11, Seite 47]. In der Praxis richtet sich dieser Service meist an Entwickler. Der Vorteil gegenüber IaaS ist ein verminderter Wartungsaufwand. Das bedeutet, dass es nicht mehr notwendig ist Software auf einem virtuellen Server zu aktualisieren oder eine Firewall auf selbigem zu konfigurieren. Ähnlich wie bei IaaS ist auch dieser Service leicht skalierbar und kann flexibel an die Bedürfnisse eines Unternehmens angepasst werden.

Prominente Vertreter für diese Art von Services sind Microsoft Azure<sup>9</sup> und Google App Engine<sup>10</sup>. Beide verfügen über ein ähnliches Angebot und ermöglichen eine stundenweise Nutzung und Bezahlung.

---

<sup>9</sup><http://www.windowsazure.com/de-de>

<sup>10</sup><https://cloud.google.com/products>

## Software as a Service

Zur SaaS-Schicht gehören Anwendungen in der Cloud, die direkt an den Endkunden adressiert sind[vgl. BKNT10, Seite 35]. Dabei wird die Software direkt angeboten, ohne dass sich der Kunde um die Wartung weiterer Ressourcen kümmern muss. Alle zur Ausführung erforderlichen Ressourcen werden durch den Anbieter gestellt und gepflegt. „Den Anspruch als Cloud-Computing-Service erfüllt es allerdings nur dann, wenn das Angebot massiv skalierbar, mehrmandantenfähig und elastisch flexibel ist“[Hö11, Seite 47].

Diese Art von Service ist die derzeit am häufigsten angebotene und diskutierte Form Cloud-services[vgl. Hö11, Seite 47]. Beispiele für diese Art des Services sind Google Maps, Google Docs oder das Customer-Relationship-Management (CRM) von salesforce<sup>11</sup>.

---

<sup>11</sup><https://www.salesforce.com/de/form/sem/landing/sales-cloud.jsp>



## 2.2 RAID Systeme

Bei einem Redundant Array of Inexpensive Disks (RAID) System werden mehrere Festplatten zu einer großen virtuellen Festplatte zusammengefasst[vgl. Man13, Seite 279 f.]. Daraus resultiert, dass mindestens zwei Festplatten für den Betrieb eines RAID-Systems erforderlich sind. Solch ein System aus mehreren Festplatten wird auch vom Betriebssystem als eine einzige logische Festplatte angesehen[vgl. Man13, Seite 279 f.]. Daten, welche in einem RAID-System gespeichert werden sollen, werden je nach verwendetem Verfahren über die physikalischen Festplatten verteilt oder redundant auf mehreren Platten gleichzeitig gespeichert[vgl. Man13, Seite 279 f.]. Insgesamt existieren sieben RAID-Systeme, welche von 0 bis 6 durchnummeriert sind[vgl. Her03, Seite 428 ff.]. Auch Kombinationen aus mehreren Verfahren sind möglich. In Anbetracht dessen, dass nur RAID-0 und RAID-1 für diese Arbeit relevant sind, da RAID-2 bis RAID-6 die Datenspeicherung von der Speicherung der Prüfinformationen trennen, werden diese fünf Verfahren nicht näher erläutert[vgl. Her03, Seite 430 ff.]. Für eine komplette Übersicht aller Verfahren sei auf die Literatur [Man13] verwiesen.

### 2.2.1 RAID-0

Ein RAID-0-System verbindet mehrere physikalische Festplatten zu einer einzigen virtualisierten Einheit. Dabei werden sogenannte Stripes auf den Platten angelegt, welche üblicherweise über Größen von 32, 64 und 128 Kilobyte (KB) verfügen. Man spricht hierbei auch von der Striping-Granularität[vgl. Man13, Seite 281 f.]. Eine Datei wird demnach zerteilt und auf alle Festplatten verteilt. Der Vorteil dieses Verfahrens liegt in dem hohen Ein-/Ausgabe (E/A)-Durchsatz[vgl. Man13, Seite 281 f.]. Jedoch ist dieses System nicht ausfallsicher, da keine Redundanz gespeichert wird. Im Sinne der Definition ist das RAID-0-System daher kein echtes RAID-System[vgl. Man13, Seite 281 f.]. Abbildung 1 zeigt dieses Verfahren auf.

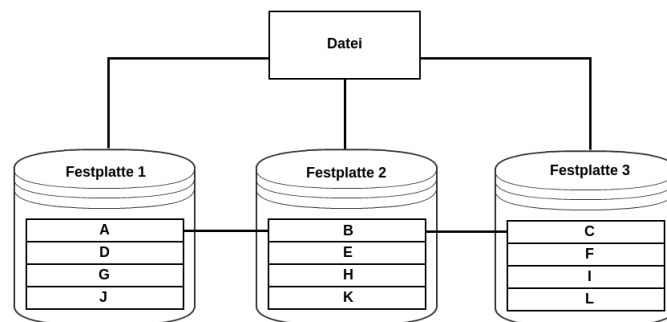


Abbildung 1: Verteilung der Teilstücke einer Datei unter Verwendung von RAID-0 (In Anlehnung an [Man08, Seite 279])

Auf jede der drei Festplatten, in der Abbildung durch Festplatte 1, Festplatte 2 und Festplatte 3 dargestellt, werden verschiedene Teilstücke einer Datei, mit den Buchstaben A bis L dargestellt, gespeichert. Daraus geht hervor, dass die Datei in mehrere Einzelstücke unterteilt und nicht redundant gespeichert wird. Sollte eine Festplatte ausfallen, ist die gesamte Datei ungültig und nicht mehr brauchbar.

### 2.2.2 RAID-1

Bei einem RAID-1-System werden alle Daten redundant auf die unterschiedlichen physikalischen Laufwerke gespeichert. Sollte der RAID-Verbund beispielsweise aus vier Festplatten bestehen, so wird eine Datei x auf allen vier Festplatten als Kopie abgelegt. Das garantiert eine größere Ausfallsicherheit, da im Falle eines Defekts einer Festplatte, die Daten noch auf weiteren Platten verfügbar sind. Dabei ist zu beachten, dass die Gesamtgröße der virtuellen Festplatte durch die kleinste verwendeten Platte festgelegt wird[vgl. Man13, Seite 281]. Die Anzahl der Festplatten kann beliebig gewählt werden. Je mehr Festplatten, desto größer die Ausfallsicherheit, wobei ein Benutzer immer den Kosten-Nutzen-Faktor abwägen muss. Das bedeutet, sowohl die Kosten für eine Anschaffung steigen, als auch die Schreibzugriffe. Ab einer gewissen Anzahl von Festplatten sinkt jedoch die Chance, dass alle zur gleichen Zeit ausfallen, auf einen vernachlässigbaren Prozentsatz. Der Nachteil bei diesem Verfahren liegt im Schreibvorgang. Dadurch, dass auf mehreren logischen Festplatten dieselbe Datei gespeichert werden muss, nimmt dies unter Umständen mehr Zeit in Anspruch als beim Beschreiben einer einzelnen Festplatte. Jedoch kann je nach Einstellung der Lesevorgang beschleunigt werden, indem einzelne Sektoren der Datei von einzelnen Festplatten gelesen werden und später zusammengeführt werden[vgl. Her03, Seite 429 f.]. Dieses Verfahren wird in Abbildung 2 aufgezeigt.

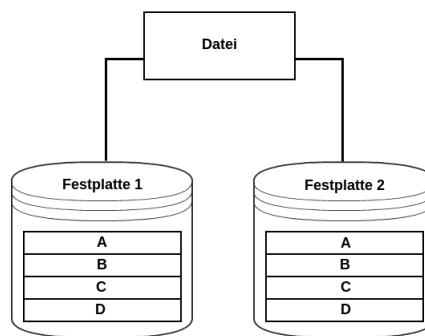


Abbildung 2: Verteilung der Teilstücke einer Datei unter Verwendung von RAID-1 (In Anlehnung an [Man08, Seite 280])

Eine Datei wird auf zwei Festplatten, in der Abbildung durch Festplatte 1 und Festplatte 2 gekennzeichnet, gespeichert. Obwohl zwei Festplatten verwendet werden, erhöht sich die Speicherkapazität nicht, da alle Teilstücke einer Datei, in der Abbildung als A bis D angegeben, gleichermaßen auf beide Festplatten verteilt werden. Der Vorteil liegt jedoch in der Ausfallsicherheit. Sollte eine der beiden Festplatten ausfallen, liegt eine Kopie der Datei immer noch auf der zweiten Festplatte, sodass der Benutzer der RAID-1-Systems weiterhin Zugriff auf seine Daten hat. Die defekte Festplatte müsste dann ausgetauscht und die Datei erneut kopiert werden.

### 2.2.3 RAID-1+0

Das RAID-1+0 System, welches auch als RAID-10 bezeichnet wird, kombiniert die Vorteile von RAID-0 und RAID-1-Systemen. Für dieses Verfahren werden mindestens vier physikalische Festplatten benötigt[vgl. Man13, Seite 282]. Jeweils 2 Platten werden dann als ein

RAID-0-Verbund zusammengeschlossen und die zwei daraus resultierenden logischen Festplatten als RAID-1-Verbund[vgl. Man13, Seite 282]. Somit ist eine Ausfallsicherheit gegeben, da Daten redundant gespeichert werden[vgl. Man13, Seite 282]. Diese Verknüpfung der beiden RAID-Systeme wird in Abbildung 3 aufgezeigt.

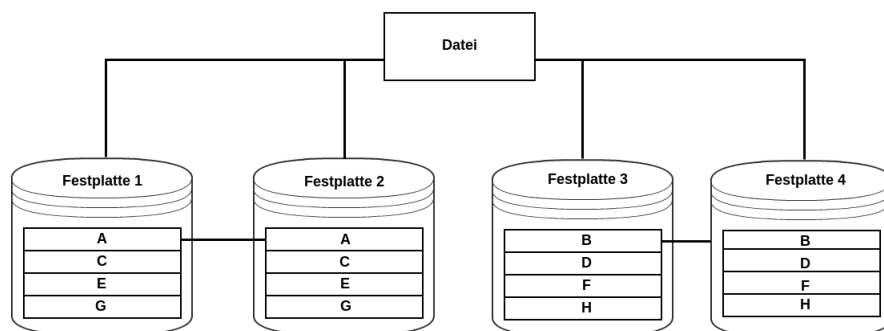


Abbildung 3: Verteilung der Teilstücke einer Datei unter Verwendung von RAID-1+0 (In Anlehnung an [Man08, Seite 281])

Die Datei wird einerseits, wie bei einem RAID-0-System, geteilt und der Speicherplatz somit vergrößert, andererseits werden die Daten auch redundant gespeichert, wie in einem RAID-1-System. Das Prinzip eines RAID-1-Systems erkennt man in der Abbildung beispielhaft an Festplatte 2 und Festplatte 3. Auch hier wird die Datei in Teilstücke von A bis H zerteilt und gespeichert. Weiterhin kann man die redundante Speicherung eines RAID-1-Systems an Festplatte 1 und Festplatte 2 erkennen. Diese speichern beide dieselben Dateiinformatoren, was bedeutet, dass bei einem Ausfall einer dieser Platten die Datei trotzdem verfügbar ist. Somit sind die Vorteile beider Systeme in einem vereint. Ein Benutzer vergrößert seinen Speicherplatz und erhält zugleich eine größere Ausfallsicherheit.

## 2.3 Kryptographie

In diesem Abschnitt wird auf kryptographische Grundlagen eingegangen, welche bei der Umsetzung des zu entwickelnden Systems relevant sind. Dazu werden zuerst Zufallszahlen und deren Besonderheiten aufgezeigt. Nachfolgend wird ein kurzer Überblick über verschiedene Verschlüsselungsverfahren, sowie deren Vor- und Nachteile herausgearbeitet. Abschließend werden Hashfunktionen erklärt und ein erweitertes Verfahren zur Nutzung selbiger, welches als Salt bezeichnet wird, erarbeitet.

### 2.3.1 Zufallszahlen

Zufallszahlen finden in vielen Computerprogrammen Anwendung. So können zufällige Laufwege von Figuren in einem Computerspiel berechnet, eine Playlist von Musiktiteln zufällig durchlaufen oder Bilder in eine Diashow in einer zufälligen Reihenfolge angezeigt werden[vgl. Lin02, Seite 62]. Dabei werden die Zufallszahlen von einem Zufallsgenerator erzeugt. Jedoch ist der Zufall „auf Tastendruck“ nicht so leicht zu realisieren[vgl. Lin02, Seite 62]. Aus diesem Grund spricht man auch von Pseudozufallsfolgen.

#### Pseudozufallsfolgen

Das Problem bei Zufallszahlen ist, dass nicht ohne Weiteres ein „echter Zufall“ erzeugt werden kann[vgl. Lin02, Seite 62]. Die Zahlenfolge wird von einem Algorithmus erzeugt, welcher bei gleichem Ausgangswert, auch das gleiche Ergebnis liefert[vgl. Lin02, Seite 62]. Daher spricht man in diesem Zusammenhang von „Pseudozufallsfolgen“. Diese reichen in vielen Fällen aus, beispielsweise bei den genannten Anwendungsbeispielen, bieten jedoch für die Kryptographie eine nur unzureichende Sicherheitsmöglichkeit[vgl. Lin02, Seite 62].

#### Kryptographisch sichere Zufallsfolgen

Die kryptographisch sicheren Zufallsfolgen unterscheiden sich von den Pseudozufallsfolgen. Diese werden immer noch von einem Zufallsgenerator erzeugt und unterliegen auch einem Algorithmus, jedoch sind sie an bestimmte Kriterien gebunden[vgl. Lin02, Seite 63 ff.]. „Demgemäß dürfen Zufallsfolgen mit realistischem Aufwand nicht vorhersagbar sein. Das bedeutet, daß selbst bei Kenntnis des Verfahrens, wie die Zufallsfolge erzeugt wurde, keinerlei Möglichkeit bestehen sollte, dieselbe Folge zu generieren“[Lin02, Seite 63]. Dazu werden unterschiedliche Verfahren verwendet. So können beispielsweise die Systemzeit, ein Mausbewegungsmuster oder Tastatureingaben verwendet werden. Ferner kann auch auf spezielle Hardware zurück gegriffen werden, welche physikalische Messwerte benutzt um diese in die Generierung mit einfließen zu lassen[vgl. Lin02, Seite 63 ff.].

### 2.3.2 Symmetrische Verschlüsselungsverfahren

Ein Verschlüsselungsverfahren wandelt einen Klartext in einen für Menschen nicht lesbaren Text um[vgl. Lin02, Seite 15]. Ziel ist es, eine Nachricht von A nach B zu übertragen, ohne das Unbefugte diese lesen können. Um einen Text zu verschlüsseln, wird ein Schlüssel, oder auch

Passphrase, verwendet. Wenn sowohl zur Chiffrierung, als auch zur Dechiffrierung ein und derselbe Schlüssel verwendet wird, dann spricht man von symmetrischen Verschlüsselungsverfahren[vgl. Lin02, Seite 69]. Dieser Schlüssel muss sowohl dem Sender als auch dem Empfänger vorliegen, sodass dieser zuvor ausgetauscht werden muss. Darin besteht auch das größte Sicherheitsrisiko, da ein Angreifer beispielsweise den Schlüssel beim Übertragen abfangen oder sich als Empfänger tarnen könnte[vgl. Lin02, Seite 69]. Um dieses Konzept zu umgehen wurde, 1978 das asymmetrische Verschlüsselungsverfahren RSA, welches nach den Anfangsbuchstaben der Entwickler Rivest, Shamir und Adleman benannt wurde, entwickelt[vgl. Lin02, Seite 115]. Dieses verwendet zum Chiffrieren und zum Dechiffrieren einen eigenen Schlüssel. Es wird auch als Public-Key-Verschlüsselung bezeichnet[vgl. Lin02, Seite 110]. Da jedoch letzteres Verfahren im Rahmen dieser Arbeit keine Relevanz hat, wird nur auf symmetrische Verfahren detailliert eingegangen.

## **Blockchiffren**

Bei dem Blockchiffrenverfahren werden sowohl Klartext als auch der Chiffretext in einzelne Blöcke fester Länge unterteilt, welche in eigenen Vorgängen unabhängig von den anderen Blöcken verschlüsselt beziehungsweise entschlüsselt werden[vgl. Lin02, Seite 72]. Dabei kann die Blocklänge, je nach verwendetem Verfahren, variieren[vgl. Lin02, Seite 72]. Sollte ein Block nicht die volle Länge besitzen, so wird dieser mit den fehlenden Bits aufgefüllt, was als Padding bezeichnet wird[vgl. Lin02, Seite 72]. Die Chiffrierung eines Blocks wird dabei als Runde bezeichnet, sodass ein Verfahren mehrere Runden durchlaufen muss. Die Anzahl der Runden ist dabei ebenfalls von dem verwendeten Verfahren abhängig.

## **DES**

Der Data Encryption Standard (DES) wurde 1974 von IBM veröffentlicht und 1977 zum offiziellen Verschlüsselungsstandard erklärt. Sowohl die Blocklänge, als auch die Schlüssellänge des DES betragen 64 Bit, wobei bei der Schlüssellänge 8 Bit als Prüfsumme verwendet werden und somit real nur 56 Bit zur Verfügung stehen[vgl. Lin02, Seite 78]. Somit ergeben sich  $2^{56}$  mögliche Schlüssel, das entspricht 72 Milliarden Schlüsseln[vgl. Lin02, Seite 80]. Das größte Sicherheitsrisiko liegt dennoch in der Schlüssellänge. Hochleistungsrechner oder auch Botnetze können eine solche Schlüssellänge mit Brute-force-Attacken in wenigen Stunden entschlüsseln[vgl. Lin02, Seite 80]. Daher gilt DES heutzutage nicht mehr als sicher. Aus diesem Grund wurde der Triple-DES Algorithmus eingeführt. Selbiger wendet lediglich DES drei mal auf eine Nachricht an[vgl. Lin02, Seite 82]. Dadurch steigt die Schlüssellänge, je nach Modus, entweder auf 112 Bit bei zwei verschiedenen Schlüsseln oder auf 168 Bit bei drei verschiedenen Schlüsseln[vgl. Lin02, Seite 82]. Der Aufwand für einen potenziellen Angreifer steigt dabei erheblich, was aus der Verwendung von  $2^{112}$  beziehungsweise  $2^{168}$  möglichen Schlüsseln resultiert[vgl. Lin02, Seite 82]. Lediglich die Geschwindigkeit ist der große Nachteil dieses Verfahrens. Es gibt mehrere Algorithmen, die schneller arbeiten, jedoch als genauso oder sogar sicher als Triple-DES gelten.

## AES

Im Jahr 2000 wurde der Advanced Encryption Standard (AES) als Nachfolger von DES eingeführt[vgl. Lin02, Seite 90]. AES verwendet den Rijndael-Algorithmus, der von Joan Daemen und Vincent Rijmen, im Rahmen einer internationalen Ausschreibung zur Einreichung von Vorschlägen für den AES im Jahr 1997, entwickelt wurde[vgl. Lin02, Seite 89]. Dieser arbeitet mit 128-, 192- oder 256-Bit-Blöcken und mit einer Schlüssellänge von 128, 192 oder 256 Bit[vgl. Lin02, Seite 89]. Die Rundenanzahl basiert sowohl auf der Block-, als auch auf der Schlüssellänge. Bis heute gilt dieses Verfahren als sicher[vgl. Lin02, Seite 94]. Für die Beschreibung der detaillierten Funktionsweise von AES sei auf die Literatur [SPS11, Seite 81 ff.] verwiesen.

## Blowfish

Der Blowfish-Algorithmus wurde im Jahr 1994 von Bruce Schneier veröffentlicht[vgl. Lin02, Seite 96]. Er arbeitet wesentlich schneller als DES und verwendet ausschließlich einfache Operationen wie Addition und XOR[vgl. Lin02, Seite 96]. Die Schlüssellänge hingegen ist variabel und beträgt zwischen 32 und 448 Bit[vgl. Lin02, Seite 96]. „Das Verfahren basiert auf zwei Teilen, einer Schlüsselexpansion, die den bis zu 448 Bit großen Schlüssel in verschiedene Teilschlüssel [...] umwandelt, die zusammen 4168 Bit ergeben. Die eigentliche Chiffrierung wird in einer Funktion F in sechzehn Runden vollzogen. Die einzelnen Runden bestehen aus einer schlüssel- und datenabhängigen Substitution und einer schlüsselabhängigen Permutation“[Lin02, Seite 97]. Für die detaillierte Funktionsweise des Blowfishalgorithmus sei auf die Literatur [Lin02] verwiesen. Blowfish ist frei zugänglich und steht damit für kommerzielle und nicht-kommerzielle Anwendungen zur Verfügung. Bis heute gibt es keine nennenswerten Schwachstellen in der Verschlüsselung[vgl. Lin02, Seite 100].

## Twofish

Twofish ist eine Weiterentwicklung des Blowfish-Algorithmus und wurde ebenfalls von Bruce Schneier, von seiner Firma Counterpane Internet Security, veröffentlicht.[vgl. Lin02, Seite 100] Das Verfahren war eines von fünf beim Ausscheid zur Bestimmung des Nachfolgers von DES[vgl. Lin02, Seite 100]. Jedoch unterlag er, genau wie Blowfish, dem Rijndael-Algorithmus in direkten Geschwindigkeitsvergleich. Genau wie Blowfish ist er frei verfügbar, was ihn sowohl für nicht-kommerzielle als auch für kommerzielle Anwendungen prädestiniert. Die Schlüssellänge beträgt 128, 192 oder 256 Bit, bei einer Blocklänge von 128 Bit und 16 Runden. Auch dieser Algorithmus zählt bis heute als sicher[vgl. Lin02, Seite 100 f.]. Er wird in Programmen wie TrueCrypt<sup>12</sup> oder auch KeePass<sup>13</sup> verwendet.

### 2.3.3 Hashfunktionen

Hashfunktionen zählen zur symmetrischen Kryptographie. Sie erzeugen aus einer Nachricht beliebiger Länge eine Prüfsumme, oftmals auch als Hashsumme oder Hashwert bezeichnet, fester Länge, meist 128 oder 160 Bit lang[vgl. Lin02, Seite 127]. Im Gegensatz zur Chiffrierung,

---

<sup>12</sup><http://www.truecrypt.com>

<sup>13</sup><http://www.keepass.com>

darf aus einem Hashwert die originale Nachricht nicht wiederhergestellt werden können[vgl. Lin02, Seite 127]. Dieses Verfahren nennt man Einweg-Hashfunktionen. Weiterhin müssen Hashfunktionen kollisionsresistent sein. Das bedeutet, dass es nahezu unmöglich sein muss, zwei unterschiedliche Nachrichten mit gleichem Hashwerten zu finden[vgl. Lin02, Seite 127]. Dies wiederum schließt nicht aus, dass es diese gibt. Nahezu unmöglich bedeutet dabei, „dass es weder mit heutigen Computern, noch mit Rechnern aus der nahen Zukunft möglich sein soll, dies in einem sinnvollen Zeitrahmen zu berechnen“[Sch10, Seite 11 f.]. Hashfunktionen werden eingesetzt, um Manipulationssicherheit zu gewährleisten. Anschaulich gesprochen bedeutet das, dass im ersten Schritt die Prüfsumme einer Nachricht erstellt wird. Daraufhin wird die eigentliche Nachricht vom Sender verschickt. Der Empfänger berechnet beim Empfangen erneut die Prüfsumme und vergleicht diese dann mit der ersten. Dies setzt voraus, dass der Empfänger die originale Prüfsumme zuvor erhalten hat. Sollten beide Werte übereinstimmen, kann der Empfänger davon ausgehen, dass die Nachricht nicht verändert wurde. Wenn die Werte hingegen nicht übereinstimmen, kann die Nachricht als kompromittiert angesehen werden. Beispiele dafür sind die Prüfsumme in einem TCP/IP Header oder Session-IDs in Webanwendungen[vgl. Sch10, Seite 11 ff.].

## MD5

Das Message-Digest Algorithm 5 (MD5) Verfahren erzeugt aus einer Nachricht mit beliebiger Länge eine 128 Bit Prüfsumme[vgl. Lin02, Seite 130]. Dieses Verfahren wurde 1991 von Ronald L. Rivest am Massachusetts Institute of Technology (MIT), entwickelt[vgl. Riv92]. „Das Verfahren verarbeitet Blöcke mit einer Länge von 512 Bit und generiert einen 128-Bit-Hashwert“[vgl. Lin02, Seite 130]. Jedoch gilt MD5 als nur bedingt sicher, nachdem 1996 eine Schwachstelle in der Kompressionsfunktion gefunden wurde[vgl. Lin02, Seite 132]. Hans Dobbertin zeige daraufhin auf, wie Kollisionen berechnet werden können[vgl. Lin02, Seite 132]. Dies widerspricht der Kollisionsresistenz von Hashfunktionen. In der Praxis wird von der Nutzung abgeraten[vgl. Lin02, Seite 132].

## SHA

Die Abkürzung SHA steht für Secure-Hash-Algorithmus und wurde 1993 vom National Institute of Standards and Technology (NIST) in Zusammenarbeit mit der National Security Agency (NSA) entwickelt[vgl. Lin02, Seite 131]. Mit Hilfe dieses Verfahrens werden 160 Bit lange Prüfsummen erzeugt[vgl. Lin02, Seite 131]. Ähnlich wie auch MD5 gilt der Secure-Hash-Algorithm (SHA) als nicht mehr sicher[vgl. Lin02, Seite 132]. Daher wurde 2005 die SHA-2-Familie veröffentlicht. Diese wird in SHA-224, SHA-256, SHA-384 und SHA-512 unterteilt. Die Nummer steht dabei für die Länge der resultierenden Prüfsumme. SHA-2 zählt bis heute als sicher. Jedoch wurde im Jahr 2008 ein internationaler Wettbewerb gestartet, wo der Nachfolger SHA-3 festgelegt werden sollte. Im Jahr 2012 wurden Guido Bertoni, Joan Daemen, Michaël Peeters und Gilles Van Assche mit dem von ihnen entwickelten Verfahren, namens „Keccak“, als Sieger bekannt gegeben<sup>14</sup>.

---

<sup>14</sup><http://www.nist.gov/itl/csd/sha-100212.cfm>

## Salt

Prüfsummen spielen auch bei der Speicherung von Passwörtern eine große Rolle. So sollten diese immer als eine Prüfsumme  $p$ , welche mit einer Einweg-Hashfunktion  $H$  erzeugt wurde, gespeichert werden. Das erschwert einem potentiellen Angreifer die Ermittlung des originalen Passworts  $x$ . Sollte er jedoch Kenntnisse über das verwendete Hashverfahren haben, so kann er mit sogenannten Rainbowtables die Zeit zum Aufspüren von  $x$  senken. Rainbowtables bestehen meist aus einfachen Key-Value-Pairs. Der Key gibt dabei die unter Verwendung von  $H$  resultierende Prüfsumme und der Value das originale Passwort an. Der Angreifer muss also lediglich  $p$  mit den Prüfsummen aus der Rainbowtable vergleichen und kann somit  $x$  ermitteln. Aus diesem Grund wurde die Verwendung eines Salt entwickelt. Bei diesem Verfahren wird der Aufwand für den Angreifer nochmals erhöht. Dazu wird neben dem  $p$ , eine weitere Prüfsumme  $s$  erstellt. Diese wird als Salt bezeichnet. In der einfachsten Form wird  $H(x + s) = p$  erzeugt. Daraus resultiert, dass der Angreifer, bevor er  $p$  mit seiner Rainbowtable vergleichen kann, eine neue Table unter Verwendung von  $s$  erzeugen muss, um diese dann zu durchlaufen. Bei einem einzelnen Passwort erhöht dieses Verfahren den Aufwand nur geringfügig, jedoch wird bei großen Benutzer-Datenbanken der Aufwand erheblich erhöht. Voraussetzung dafür ist, dass jeder Benutzer seinen eigenen Salt erhält. Darüber hinaus gibt es noch Abwandlungen dieses Verfahrens, wo der Salt nicht an das Ende des Passwortes geschrieben wird, sondern geteilt und an den Anfang und das Ende gestellt wird. Somit wird der Aufwand zum Ermitteln von  $x$  erneut vergrößert, jedoch nur solange der Angreifer keine Kenntnis über die Art der Verwendung von  $s$  hat.



## 2.4 OAuth

Wenn ein Benutzer einer Anwendung den Zugriff auf seine Daten erlauben möchte, die von einer weiteren Anwendung verwaltet wird, so kann er Dank des offenen OAuth Protokolls<sup>15</sup>, dieses ohne seine Zugangsdaten preiszugeben. Dieses Konzept wird besonders bei APIs eingesetzt. Prominente Beispiele dafür sind Facebook und Twitter. Diese ermöglichen es, dass Entwickler eigene Applications (Apps) in das System integrieren können, welche Zugriff auf Benutzerdaten erhalten. Meldet sich ein Benutzer bei einer Anwendung eines Drittanbieters an, so wird dieser auf die Webseite des eigentlichen Anbieters, also beispielsweise Twitter weitergeleitet, wo er sich anmelden muss. Twitter schickt dann einen benutzerspezifischen Schlüssel an die Anwendung, mit dem diese dann Zugriff auf die Benutzerdaten erhält. In vielen APIs wird der Benutzer zusätzlich noch darauf hingewiesen, auf welche Daten die Anwendung Zugriff erhält. Dies ist jedoch nicht Bestandteil der OAuth Spezifikation.

### 2.4.1 Historischer Hintergrund

Bevor OAuth entwickelt wurde, arbeiteten bereits einige Firmen, wie beispielsweise Twitter<sup>16</sup> und Ma.gnolia<sup>17</sup>, an der Implementierung des OpenID<sup>18</sup> Protokolls in ihren Webanwendungen. Jedoch war dieses Verfahren nicht für autorisierungspflichtige Schnittstellen ausgelegt, da ein Benutzer, der OpenID verwendet, kein Passwort mehr hat. Dieses ist allerdings zwingend notwendig, da er sich sonst nicht an der Schnittstelle anmelden kann. Es existierten bereits einige alternative Verfahren, wie beispielsweise Flickr Auth, Google AuthSub oder Yahoo! BBAuth, jedoch kein definierter Standard. Daraufhin wurde im Dezember 2006 ein Treffen zwischen diversen Entwicklern und David Recordon, einem Entwickler von OpenID, organisiert, bei dem über einen Authentifizierungsstandard diskutiert werden sollte. Aus dem Treffen entstand OpenAuth, welche später zu OAuth umbenannt wurde. Im Oktober 2007 wurde die finale OAuth Core 1.0 Spezifikation veröffentlicht[vgl. Ham11a].

Knapp fünf Jahre später, im Oktober 2012 wurde die OAuth 2.0 Spezifikation herausgebracht[vgl. Har12]. An diesem Verfahren gab es jedoch viel auszusetzen, was Eran Hammer<sup>19</sup>, einen Mitbegründer der OAuth 1.0 und OAuth 2.0 Spezifikation, dazu veranlasste, von seinem Posten als Vorsitzender des OAuth Komitees, zurückzutreten[vgl. Ham10a]. Er selbst sagt, dass OAuth 2.0 im Vergleich zum OAuth 1.0 „viel komplexer, weniger interoperabel, weniger nützlich, unvollständiger und vor allem weniger sicher“[Ham12] ist. Dennoch wird diese Methode in der Praxis verwendet und löst nach und nach OAuth 1.0 ab.

### 2.4.2 OAuth 1.0

In der Spezifikation von OAuth 1.0 werden drei Rollen festgelegt. Es gibt den »Consumer«, welcher dem Client oder auch einer Anwendung entspricht, der »Service Provider«, welcher der Server ist, beispielsweise Twitter, und der »User«, der auch als Resource Owner bezeichnet wird[vgl. Ham10b]. In einem klassischen Client-Server Authentifizierungsverfahren muss

---

<sup>15</sup><http://oauth.net/documentation>

<sup>16</sup><http://www.twitter.com>

<sup>17</sup><http://gnolia.com>

<sup>18</sup><http://openid.net>

<sup>19</sup><http://hueniverse.com/author/eran>

der User seine Zugangsdaten, welche meist aus einer Email oder einem Benutzernamen und einem Passwort bestehen, beim Server eingeben[vgl. Ham11a]. Dabei ist es dem Server egal, von wem er diese Daten erhält, unabhängig davon, ob es der User selbst ist oder ein anderer Anbieter[vgl. Ham11a]. Sobald ein Drittanbieter an diesem Prozess beteiligt ist, hat dieses Verfahren einen gewaltigen Nachteil. Der Drittanbieter bekommt Kenntnis über die eigentlich geheimen Zugangsdaten des Users. Durch die Verwendung von OAuth bleiben diese Zugangsdaten jedoch beim Server und der Client interagiert nur noch mit Hilfe eines Tokens mit dem Server.

Die Anzahl der eingebundenen Anbieter wird durch eine Anzahl von »Füßen« beschrieben. So wäre beispielsweise der letztgenannte Fall eine 3-beinige Authentifizierung, im Englischen 3-legged Authentication genannt. Weitere Legs werden unterschiedlich definiert, wobei generell gilt, dass sobald ein Anbieter Zugriff auf die Nutzerdaten beansprucht, dieser als ein weiteres Leg gezählt wird[vgl. Ham11a].

Weiterhin gibt es drei Arten von Token, also einzigartige Schlüssel, in der OAuth Spezifikation. Diese werden als »client credentials«, »temporary credentials« und »token credentials« bezeichnet[vgl. Ham11a]. Zu den »client credentials« zählt der »consumer key« und das »consumer secret«[vgl. Ham11a]. Diese kennt der Consumer und benötigt diese, um dem Service Provider mitzuteilen, welche Anwendung Zugriff auf Benutzerdaten haben möchte.

Der »request token« und das »request secret« gehören zu den »temporary credentials«, welche der Consumer benutzt, um den Service Provider aufzufordern den User zu verifizieren[vgl. Ham11a].

Als »token credentials« werden der »access token« und das »access secret« bezeichnet[vgl. Ham11a]. Diese dienen nach einer erfolgreichen Authentifizierung fortan als Erkennungsschlüssel für den User. Ein Consumer kann mit den Token Benutzerdaten anfordern oder diese bearbeiten.

Abbildung 4 zeigt das Verfahren schematisch auf.

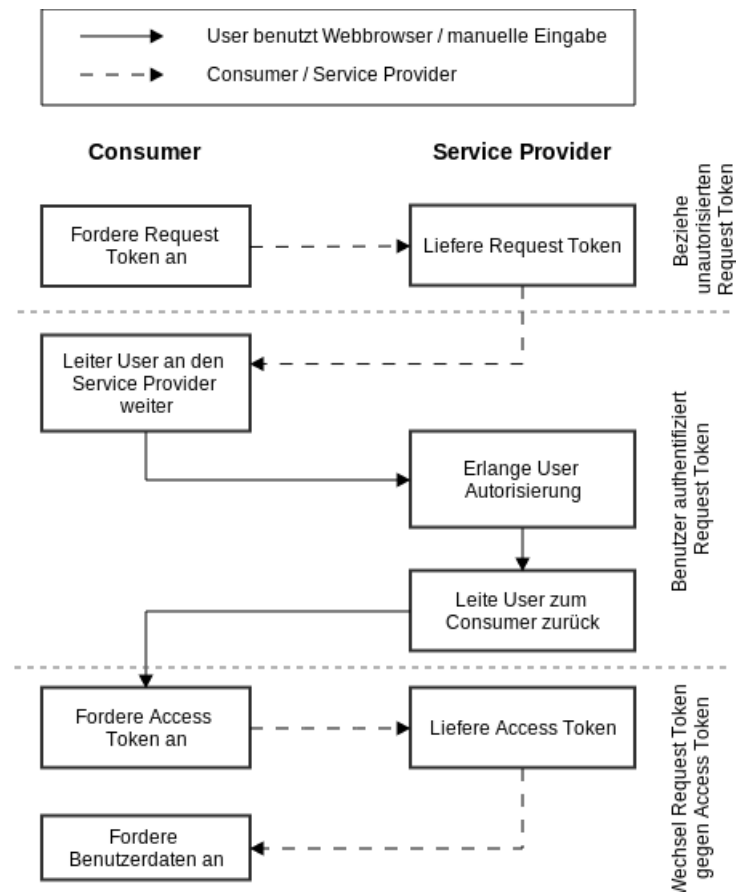


Abbildung 4: Workflow des OAuth 1.0 Authentifizierungsprozesses (In Anlehnung an [OAu07])

In der Praxis würde sich der User beispielsweise für ein Textverarbeitungsprogramm von Google, welcher in diesem Fall der Consumer ist, interessieren. Die Dateien, die er bearbeiten möchte, liegen jedoch bei Dropbox, dem Service Provider. Der User möchte nun Google für den Zugriff auf seine Daten bei Dropbox berechtigen. Dropbox verwendet eine offene API, welche OAuth unterstützt und Google hat bereits bei dieser eine Entwickleranwendung angelegt. Nach dem Anlegen hat Google die »client credentials« für seine Anwendung erhalten.

Wenn der User nun Google dazu berechtigen möchte, auf seine Benutzerdaten zugreifen zu können, muss er dazu einen Button auf der Webseite von Google betätigen. Daraufhin werden zwei Vorgänge durchgeführt. Zuerst schickt Google seine »client credentials« an Dropbox und erhält, insofern diese Gültig sind, die »temporary credentials« zurück. Sobald Google diese erhalten hat, wird der User auf die Loginseite von Dropbox weitergeleitet. Dort gibt er seine Zugangsdaten ein, um seine Authentizität zu beweisen. Nachdem er eingeloggt ist, bekommt er eine Bestätigungsseite angezeigt, auf der er Google Zugriff auf seine Daten gewähren kann. Hat er dem zugestimmt, wird er zu Google zurückgeleitet. Google erhält daraufhin den »request token« zurück, welcher ihm aufzeigt, dass ihm der User den Zugriff gewährt hat. Abschließend schickt Google seinen »consumer key« und den »request token« an Dropbox und erhält die »token credentials«. Mit diesen kann Google fortan die Benutzerdaten von Dropbox abrufen, verändern und die gewünschten Textdateien des Users in seinem Service einbinden. Im OAuth 1.0 Standard ist jedoch kein Parameter angegeben, welcher den Gültigkeitszeitraum der »token

credentials« festlegt. Somit sind diese für immer gültig oder entsprechend der Implementierung des Service Providers werden diese nach einem gewissen Zeitraum verworfen. Jedoch gibt es keine Vereinheitlichung in diesem Punkt.

### 2.4.3 OAuth 1.0A

Nachdem im Jahr 2009 eine Sicherheitslücke in der OAuth 1.0 Methode entdeckt wurde, haben Google und Yahoo! eine weitere Revision von OAuth 1.0, welche später als OAuth 1.0A veröffentlicht wurde, entwickelt[vgl. OAu09]. Der Angriff auf das Authentifizierungsverfahren, welcher als »session fixation attack« bezeichnet wird, hat folgenden Ablauf.

Der Angreifer Mallory möchte ein Textdokument aus seiner Dropbox mit dem Google Textbearbeitungsprogramm bearbeiten. Er initiiert dazu den OAuth Autorisierungsprozess bei Google, unterbindet jedoch die Weiterleitung zu Dropbox und speichert sich den Link dahin ab. Dieser enthält die »client credentials« und die URL zur Authentifizierungsseite. Nachfolgend bringt er einen Benutzer Alice dazu, diesen Link anzuklicken.

Alice führt den von Mallory gestarteten Autorisierungsprozess fort und wird zu Dropbox geleitet. Dort gibt sie ihre Benutzerdaten ein und akzeptiert die Berechtigung der Google App Zugriff auf ihre Daten zu haben. Sie bekommt jedoch nicht mit, dass ein Angriff durchgeführt wird, da sie sich beim gültigen Serviceprovider befindet und auch die korrekte App angezeigt bekommt. Nachdem Alice zurück zur Applikation geleitet wurde, führt Mallory seinen Autorisierungsprozess fort, indem er die Weiterleitung von Dropbox zurück zu Google nachbaut oder diese bei Alice manipuliert. Von diesem Zeitpunkt an ist der Dropbox-Account von Alice mit der Google App von Mallory verbunden.

Das ganze Verfahren ähnelt einem Pishingverfahren und setzt auf Social Engineering, was bedeutet, dass das Vertrauen und die Gewohnheit eines unachtsamen Benutzers ausgenutzt wird und dieser einem Link aus einer nicht vertrauenswürdigen Quelle folgt.

Um diese Sicherheitslücke zu schließen, wurde als vorzeitige Lösung ein Warnhinweis vom Serviceprovider, also im Beispiel Dropbox, angezeigt, wenn der Benutzer von einer Uniform Resource Locator (URL) weitergeleitet wurde, die nicht mit der URL des App Betreibers übereinstimmt, im Beispiel Google[vgl. Ham09]. Diese Variante hat sich jedoch als unzureichend herausgestellt, sodass das OAuth 1.0A Protokoll entwickelt wurde.

In OAuth 1.0A wurden drei wesentliche Änderungen durchgeführt. Wenn der Consumer die »temporary credentials« beim Service Provider anfordert, muss er eine Callback URL angeben[vgl. OAu09]. Dieser Parameter existierte bereits in der OAuth 1.0 Spezifikation, war dort aber optional und wurde erst beim Weiterleiten zum Service Provider mitgesendet, wenn die »session fixation attack« bereits gestartet ist. Nach Abschluss der Autorisierung beim Service Provider wird der User an diese URL weitergeleitet.

Weiterhin wurde der Parameter »oauth\_callback\_confirmed« eingeführt, der den Wert true hat und bei der Anfrage des »request token« an den Consumer geschickt wird[vgl. OAu09]. Dieser dient lediglich zur Erkennung einer OAuth 1.0A Unterstützung Seitens des Service Providers.

Die letzte Änderung betrifft die Weiterleitung des Users vom Service Provider zum Consumer. Hierbei wird der Parameter »oauth\_verifier« implementiert, der vom Consumer entgegengenommen und bei der Anfrage der »token credentials« mitgeschickt werden muss[vgl. OAu09].

Diese drei Änderungen haben die »session fixation attack« ungültig gemacht. Das OAuth Komitee hat darüber hinaus jedem Anbieter dringendst geraten sein System auf OAuth 1.0A zu aktualisieren[vgl. Ham09].

#### 2.4.4 OAuth 2.0

OAuth 2.0 wurde im Oktober 2012 veröffentlicht und ist ein komplett neues Protokoll, welches nicht abwärtskompatibel zu OAuth 1.0 oder OAuth 1.0A ist[vgl. Ham10a]. Es gibt wesentliche Veränderung im Workflow und in der Kommunikation mit dem Server. Dabei sind drei Unterschiede als besonders prägnant zu erachten. Jegliche Verbindungen erfolgen über Secure Socket Layer-Technologie (SSL) und es muss keine eigene Signatur, wie bei den vorherigen OAuth Protokollen, erzeugt und geprüft werden[vgl. Har12].

Des Weiteren existiert nur noch ein Token in den »token credentials«. Dieser wird »access\_token« genannt und dient fortan zur Authentifizierung des Users[vgl. Har12]. Ein optionaler »refresh\_token« kann vom Service Provider an den Consumer geschickt werden, welcher oftmals mit einem Parameter »expires\_in« übergeben wird[vgl. Har12]. Letzterer gibt in Millisekunden an, wie lange der »access\_token« gültig ist[vgl. Har12]. Sollte dieser abgelaufen sein, so kann ohne das Eingreifen des Users mithilfe des »refresh\_token« ein Neuer angefordert werden. Ein weiterer Vorteil dieses Prinzips ist, dass die Laufzeit der »token credentials« explizit festgelegt wird und diese nicht wie bei OAuth 1.0 einen unbegrenzten Verwendungszeitraum ermöglichen. Häufig wird dieser Zeitraum auf 3600 Millisekunden, also eine Stunde, festgesetzt. Das OAuth Komitee empfiehlt dringend die Implementierung des »refresh\_token« und des »expires\_in« Parameters, jedoch sind beide optional, womit die Entscheidung beim Service Provider liegt, diese zu verwenden oder nicht[vgl. Har12].

Weiterhin gibt es vier Rollen im OAuth 2.0 Standard. Es gibt den »resource owner«, welcher dem User entspricht, also einem Benutzer einer Anwendung[vgl. Har12]. Ebenfalls gleich geblieben ist der Client, welcher auch als Consumer bezeichnet wird[vgl. Har12]. Neu ist die Unterteilung in einem »resource server« und einem »authorization server«[vgl. Har12]. Der »resource server« hält die Benutzerdaten vor, während der »authorization server« die »credentials«, also jegliche Token speichert[vgl. Har12]. Diese Unterteilung erleichtert die Implementierung des Protokolls auf Seiten des Server Providers, der meist beide Rollen zugleich einnimmt. Lediglich die Domain kann voneinander abweichen.

Der Workflow von OAuth 2.0 hat sich in der Praxis wie folgt verändert. Wie bereits in den vorherigen Spezifikationen muss der Consumer eine Entwicklerapp beim Service Provider anlegen. Dabei muss er bereits eine Callback URL definieren, an die der User nach dem Autorisierungsprozess weitergeleitet wird. Der Consumer erhält nach Abschluss der Erstellung die »client credentials«, also eine »client\_id«, sowie ein »client\_secret«. Sobald ein User einen Consumer Autorisieren möchte, Zugriff auf seine Daten zu erhalten, wird er wie gewohnt zum Service Provider weitergeleitet. Jedoch entfällt der bisherige Prozess zum Erzeugen der »temporary credentials«. Der Consumer schickt lediglich seine »client\_id« bei Weiterleiten an den Service Provider, wodurch dieser die entsprechende Entwicklerapp eruieren kann.

Beim Service Provider bekommt der Benutzer, wie zuvor, die Loginseite angezeigt und muss nach erfolgreichem Login der Entwicklerapp den Zugriff gewähren oder diesen ablehnen. Sobald er zugestimmt hat, wird er an die Callback URL des Consumers geleitet, welche bei der Erstellung der Entwicklerapp definiert wurde. Dabei wird ein Parameter mit der Bezeichnung

»code« mitgeschickt. Zum Erzeugen eines »access\_tokens« muss dieser, samt der »client\_id« an den Service Provider geschickt werden. Dieser überprüft den »code« und schickt den »access\_token«, sowie optional den »refresh\_token« und die Lebenszeit des »access\_tokens« an den Consumer zurück. Ein Beispiel für solch eine Antwort ist in Quellcode 1 zu sehen.

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json;charset=UTF-8
3 Cache-Control: no-store
4 Pragma: no-cache
5
6 {
7   "access_token":"2YotnFZFEjr1zCsicMWpAA",
8   "token_type":"example",
9   "expires_in":3600,
10  "refresh_token":"tGzv3JOkF0XG5Qx2TIKWIA",
11  "example_parameter":"example_value"
12 }
```

Quellcode 1: Aufbau eines Response vom Service Provider an den Consumer[Har12]

Der Consumer kann nun mit den erhaltenen »access\_token« auf die Benutzerdaten zugreifen. Sollte die Lebenszeit des Tokens abgelaufen sein, muss dieser unter Zuhilfenahme des »refresh\_tokens« neu angefordert werden. Das bedeutet, der Consumer macht einen Request mit dem »refresh\_token« und der »client\_id« an den Service Provider und dieser schickt, sowohl einen neuen »access\_token«, als auch einen neuen »refresh\_token« und ein »expires\_in« an die Anwendung zurück. Dieses Verfahren kann beliebig oft wiederholt werden.

# Kapitel 3

## Anforderungsanalyse

Die bisher erarbeiteten Grundlagen sollen in diesem Kapitel dazu dienen, die Anforderungen für die zu entwickelnde Anwendung zu formulieren. Dabei werden Aspekte betrachtet, die bei der konkreten Umsetzung relevant sein werden.

### 3.1 Systemanforderung

Mit CloudGrid soll ein Prototyp für eine Desktopapplikation erstellt werden, der es dem Benutzer ermöglicht, verschiedene Cloudservices zu einem einzigen System zusammenzufassen. Dieses soll auf einem RAID ähnlichen Prinzip basieren, dass bedeutet, dass Dateien redundant auf mehreren Services gespeichert werden sollen. Der Vorteil für den Benutzer liegt darin, dass ihm durch diese Vereinigung einerseits mehr Speicherplatz zur Verfügung steht, andererseits eine größere Ausfallsicherheit gewährleistet werden kann. Selbst wenn ein Anbieter seinen Dienst einstellt oder temporär nicht verfügbar ist, so ist es dem Nutzer somit noch möglich, weiterhin Zugriff auf seine Daten zu erhalten. Die gesamte Dateiverwaltung soll auf dem Client durchgeführt werden. Es wird bewusst auf einen Serverdienst verzichtet, welcher die Verwaltungslogik der verschiedenen Services übernimmt. Dadurch soll das System für den Benutzer nachvollziehbarer sein, da er sich nicht bei einem weiteren Dienst anmelden muss und ihm mehr Kontrolle über den Verbleib seiner Daten geben.

Neben dem Zusammenschluss mehrerer Cloudservices, ist das Dateihandling eine weitere zentrale Aufgabe von CloudGrid. Hierbei soll ein, vom Benutzer festgelegter Ordner auf Veränderungen überprüft werden. Dazu zählen auch Unterordner mit beliebiger hierarchischer Tiefe. Wenn eine Veränderung festgestellt wird, sollen mehrere Dateioperationen durchgeführt werden. Das bedeutet konkret, dass vor dem Upload einer Dateien, diese clientseitig komprimiert und somit der Upload beschleunigt werden soll. Um die Datensicherheit zu erhöhen, wird die Datei darüber hinaus beim Client verschlüsselt. Der Benutzer verschickt dadurch keine unverschlüsselten Dateien, welche beispielsweise durch Angriffe von einem Hacker abgefangen und ausgelesen werden könnten. Weiterhin liegt die Datei verschlüsselt beim dem Cloudanbieter, wodurch dieser keine Möglichkeit hat, den Inhalt der Datei einzusehen. Um diesen Aspekt noch zu verstärken, wird die Datei in mehrere Teilstücke fragmentiert, um diese daraufhin, wie bereits erwähnt, redundant auf verschiedenen Anbietern zu speichern. Bei sensiblen Daten, wie beispielsweise Dokumenten mit Passwörtern, trägt dieses Verfahren nur bedingt zur

Dateisicherheit bei, da weiterhin Informationen über den Inhalt gewonnen werden können. Jedoch erschwert es einem potentiellen Angreifer Einsicht in die komplette Datei zu erhalten. Wird hingegen eine Datei gelöscht, muss das ebenfalls von CloudGrid erkannt werden und entsprechend alle Dateien auf den Cloudservices gelöscht werden.

Der zu erstellende Prototyp wird darüber hinaus nur eine Synchronisierung mit einem Client unterstützen. Die Synchronisierung mit mehr als einem Client setzt ein komplexes Dateimanagement voraus, welches den Rahmen dieser Arbeit überschreiten würde.

## 3.2 Cloudanservices

Das Programm CloudGrid soll, wie bereits beschrieben, eine Verbindung mit mehreren Cloudanservices aufnehmen können. Dazu müssen zuvor relevante Dienste evaluiert werden. Dabei wurden folgende Kriterien als ausschlaggebend erachtet:

**Serviceart:** CloudGrid unterstützt ausschließlich Services, welche sich als Online-Speicher verstehen. Services wie beispielsweise Evernote oder Spotify, welche ebenfalls als Cloudservices angesehen werden, werden nicht unterstützt.

**Lizenzmodell:** Ein potenzieller Clouddienst muss über einen kostenlosen Service verfügen. Dennoch soll CloudGrid darauf ausgelegt sein, kostenpflichtige Services einbinden zu können. Auf eine ausführliche Analyse kann in dieser Arbeit aus finanzieller Sicht nicht eingegangen werden.

**Programmierschnittstelle:** Die Programmierschnittstelle, auch API genannt, des Anbieters muss für Nutzer des kostenfreien Services verfügbar sein. Einige Anbieter bieten ihre Schnittstelle nur an, wenn der Benutzer einen kostenpflichtigen Service in Anspruch nimmt. Das ist sowohl für den Nutzer nachteilig, da er, falls er den Dienst zuvor testen möchte, diesen nicht in CloudGrid einbinden kann, als auch im Rahmen dieser Arbeit finanziell nicht realisierbar. Anbieter die über keine API verfügen, werden darüber hinaus nicht vom Programm unterstützt. Bei der Umsetzung des Prototypen wird lediglich JavaScript Object Notation (JSON) als Datenformat zugelassen, da dieses von vielen Anbietern verwendet wird.

**Authentifizierungsmethoden:** Der Service muss über eine OAuth Authentifizierung verfügen. Diese kann sowohl in Version 1.0, 1.0A, als auch 2.0 vorliegen. Alle drei Versionen sollen in CloudGrid vollständig implementiert werden. Eigene Authentifizierungsverfahren sind aufwändiger zu integrieren und für den Prototypen nicht vorgesehen.

**Rechtliche Anforderungen:** Die rechtlichen Aspekte bei der Entwicklung von CloudGrid spielen eine elementare Rolle. Hierbei muss betrachtet werden, ob die AGB's der Anbieter, Programme, wie das zu entwickelnde System, einschränken oder sogar verbieten. Weiterhin muss geprüft werden, ob und inwiefern der Anbieter Rechte an den Daten erhält. Zudem wird geprüft, ob etwaige Limitierungen, wie eine bestimmte Anzahl von maximalen Requests, bei der Nutzung der API existieren.



### 3.3 Technologien

Nachdem die Rahmenbedingungen für die Cloudservices definiert wurden, soll auch auf die technologischen Anforderungen von CloudGrid eingegangen werden.

Bei der Wahl der Programmierumgebung wird besonderer Wert auf eine schnelle Bearbeitung des Filesystems gelegt. Dies ist besonders wichtig, da CloudGrid viele Dateioperationen durchführen muss und der Nutzer nicht unnötig lange auf diese warten soll. Da das Programm als Dienst im Hintergrund permanent ausgeführt wird, ist es darüber hinaus wichtig, dass zur Laufzeit nicht unnötig viele Ressourcen verbraucht werden. Nicht zuletzt soll eine plattformunabhängige Entwicklung möglich sein.

Das Graphical User Interface (GUI) soll dem Benutzer zur Verwaltung von CloudGrid dienen. Als wichtig bei der Wahl der Umgebung wird eine für den Benutzer gewohnte Umgebung erachtet. Er soll sich schnell in diese einfinden und alle relevanten Informationen erhalten. Beim ersten Start von CloudGrid erhält der Benutzer die Möglichkeit persönliche Einstellungen zu tätigen. Dazu zählt das Festlegen des zu überwachenden Ordners, Einbinden der Cloudservices und eine kurze Einführung in die Funktionalität von CloudGrid. All diese Einstellungen können später noch bearbeitet werden. Nachdem die grundlegende Initialisierung abgeschlossen ist, erhält der Benutzer die Möglichkeit Auskunft über durchgeführte Dateioperationen und dem Fortschritt des Dateiuploads zu bekommen. Darüber hinaus kann er sich Informationen zu den eingebundenen Cloudservices anzeigen lassen, wie beispielsweise die Auslastung des Speicherplatzes bei diesem Anbieter oder eine Liste der hochgeladenen Dateien.

Zusätzlich soll es Entwicklern möglich sein, die GUI anzupassen und um eigene Funktionalitäten zu erweitern.

Auch bei der Wahl der GUI ist die Plattformunabhängigkeit ein wichtiger Aspekt. Die GUI soll sich auf allen Systemen gleich verhalten und sich optisch nicht oder nur gering unterscheiden.

Alle Einstellungen von CloudGrid und Einstellungen, die vom Benutzer getätigt wurden, sollen lokal beim Client vorgehalten und nicht mit einem weiteren Dienst synchronisiert werden. Das hat den Vorteil, dass der Benutzer seine Informationen nicht an einen Drittanbieter weitergeben muss und somit eine größere Dateiintegrität gewährleistet wird. Neben den allgemeinen Einstellungen sollen auch die Dateiinformationen, wie beispielsweise die Verteilung der Teilstücke einer Datei auf den verschiedenen Cloudservices, deren Passphrase zum Entschlüsseln der Datei und die Hashwerte der originalen Datei gespeichert werden. Außerdem sollen Informationen über die einzelnen Provider gespeichert werden, wie beispielsweise die App-Token zum Authentifizieren. Zur Speicherung der Daten muss daher ein geeignetes Datenbanksystem evaluiert werden. Dieses soll sich vom Programmierer leicht in die Anwendung integrieren lassen und ohne großen initialen Aufwand eingerichtet werden können.

### 3.4 Use-Case-Analyse

In der Use-Case-Analyse sollen beispielhaft vier Fälle aufgezeigt werden, welche bei der Benutzung von CloudGrid auftreten können.

**Standardverhalten:** Der Benutzer startet CloudGrid. Daraufhin überwacht die Anwendung den zuvor ausgewählten Ordner auf Veränderungen. Nun bearbeitet der Benutzer eine Datei. CloudGrid erkennt eine Veränderung am Filesystem und zusätzlich, dass es sich dabei um eine Bearbeitung einer Datei handelt. Die Datei wird daraufhin komprimiert, gesplittet und die Teilstücke verschlüsselt. Dateiinformationen über die Teilstücke werden daraufhin in die lokale Datenbank gespeichert und die Teilstücke anschließend zu den jeweiligen Anbietern geuploaded.

**Dateiänderung vor Programmstart:** Der Benutzer bearbeitet eine Datei, bevor CloudGrid gestartet wurde. Beim Start der Anwendung wird die Ordnerüberwachung gestartet. Daraufhin werden die bestehenden Dateien auf Veränderung überprüft. Zudem wird geprüft, ob neue Dateien hinzu gekommen sind oder bestehende gelöscht wurden. Sollte eine Dateiänderung erkannt werden, werden die zuvor genannten Dateioperationen durchgeführt und die Teilstücke geuploaded.

**Dateiänderung ohne Internetverbindung:** Der Benutzer startet CloudGrid und zu diesem Zeitpunkt besteht eine Internetverbindung. Die Ordnerüberwachung wird gestartet. Daraufhin wird die Verbindung mit dem Internet getrennt. CloudGrid wird in den Offline Modus geschaltet. Das bedeutet, dass alle aktiv laufenden Dateioperationen abgebrochen werden und die Ordnerüberwachung gestoppt wird. Sobald die Internetverbindung wiederhergestellt ist, ermittelt die Anwendung alle durchgeführten Veränderungen und führt entsprechend die Dateioperationen durch.

**Teilstück wird auf Server gelöscht:** Der Benutzer löscht ein Teilstück einer Datei direkt beim Cloudservice A. CloudGrid wird vom Benutzer gestartet. Die Anwendung prüft anhand der lokalen Datenbank, ob es Veränderungen bei den Cloudservices gibt. Die Löschung des Teilstücks bei Cloudservice A wird erkannt und geprüft, bei welchem Anbieter die Datei redundant gespeichert wurde. Daraufhin wird diese von Cloudservice B heruntergeladen, beim Benutzer zwischengespeichert und zum Cloudservice A erneut hochgeladen. Sollte das Teilstück auf beiden Services nicht mehr existieren, so werden alle Teilstücke der Datei auf allen Services gelöscht. Daraufhin werden die Dateioperationen auf der lokalen Datei beim Client durchgeführt und die Teilstücke erneut geuploaded.

### 3.5 Abgrenzung zu bestehenden Systemen

Im folgenden Abschnitt sollen Abgrenzungen zu verschiedenen bestehenden Systemen aufgezeigt werden. Es werden nur Dienste betrachtet, die auf bestehende Cloudservices aufsetzen und diese um Funktionen, wie das Verschlüsseln von Dateien oder den Zusammenschluss mehrerer Dienste, erweitern.

**Boxcryptor:** Boxcryptor ist ein Programm, um Daten bei verschiedenen Cloudanbietern zu verschlüsseln. Dabei beschränkt sich dieser Dienst auf einen Anbieter. Eine Vereinigung mehrerer Dienste, wie bei CloudGrid, ist nicht gegeben. Darüber hinaus wird ein Benutzerkonto beim Anbieter benötigt. Die Dateien werden auf Benutzerseite verschlüsselt. Da jedoch der Anbieter potentiell die Möglichkeit hat, die Dateien durch seinen Dienst zu entschlüsseln, ist die Dateisicherheit nur bedingt gegeben. Der Vorteil liegt jedoch darin, dass sowohl ein Programm für die Nutzung auf Desktoprechnern verfügbar ist, als auch Apps für mobile Endgeräte wie Android und iOS. Um eine erweiterte Version von Boxcryptor verwenden zu können, muss ein kostenpflichtiger Premium Account gekauft werden. Dieser beinhaltet Funktionen wie die Nutzung von mehr als zwei Geräten oder die Verschlüsselung von Dateien auf mehr als einem Cloudservices.

**Cloudii:** Cloudii ist eine Android App, die es dem Benutzer ermöglicht, mehrere Cloud-dienste zu verwalten. Dabei ist es möglich, Daten redundant zu speichern, jedoch nicht diese zu verschlüsseln. Weiterhin ist die Anwendung ausschließlich für Android verfügbar, was Benutzer mit anderen Systemen ausschließt.

**Otixo:** Das ist eine kostenpflichtige Webapplikation, welche die Verwaltung von Dateien über mehrere Cloudanbieter ermöglicht. Dabei ist eine redundante Speicherung der Daten ebenso wenig möglich, wie die Verschlüsselung selbiger.

**Primadesk:** Auch Primadesk ist eine kostenpflichtige Webapplikation, bei der es ebenso wie bei Otixo, nicht möglich ist, Dateien zu verschlüsseln oder redundant zu speichern. Jedoch werden mobile Apps sowohl für Android als auch für iOS angeboten.

**Cloudfuze:** Cloudfuze ist eine reine Desktopapplikation, die ebenfalls kostenpflichtig ist. Auch diese Anwendung dient nur der Verwaltung mehrerer Clouddienste. Daher bietet sie einen ähnlichen Funktionsumfang wie Otixo oder auch Primadesk.

**TrueCrypt:** Neben den bisher vorgestellten Anwendungen gibt es auch lokale Lösungen zum Verschlüsseln der Daten. Eine Möglichkeit besteht darin Ordner mittels TrueCrypt zu verschlüsseln. Die Anwendung erzeugt aus den verschlüsselten Daten einen Container, welcher als virtuelle Festplatte von dem Programm selbst gemountet werden kann. Sowohl zum Verschlüsseln, als auch zum mounten, muss der Benutzer ein festgelegtes Passwort eingeben. Mit der Methode kann der Benutzer den Ordner, welcher von dem Cloudservices synchronisiert wird, Verschlüsseln und nur den Container zum Anbieter hochladen. Dadurch sind die Daten geschützt, jedoch lassen sich somit nicht mehrere Cloudanbieter zusammenfassen. Auch die Ausfallsicherheit ist bei dieser Methode nicht gewährleistet.

Darüber hinaus muss der Benutzer bei TrueCrypt zuvor die Größe des Containers bestimmen. Das bedeutet, dass selbst wenn nur eine Textdatei in einem Container liegt, die Gesamtgröße der Datei festgelegt ist. Somit ergibt sich ein Overhead und die Uploadzeit steigt, je nach gewählter Dateigröße, stark an. Ein weiterer Nachteil ist, dass auf jedem Gerät, welches dieses Verfahren verwendet, TrueCrypt installiert sein muss, um den Inhalt der Dateien betrachten

zu können. Es gibt mittlerweile auch mobile Apps, die TrueCrypt-Container entschlüsseln, jedoch ist eine Verwaltung im Web derzeit nicht möglich.

Es lässt sich festhalten, dass die vorgestellten Dienste Cloudservices um sinnvolle Funktionalitäten erweitern. Jedoch bietet keiner dieser Dienste die Funktionsvielfalt von CloudGrid. Die meisten der hier vorgestellten Dienste verbinden mehrere Services zu einem gemeinsamen, beziehungsweise ermöglichen die gleichzeitige Verwaltung mehrerer Services. Dabei wird wenig Wert auf die Datensicherheit gelegt. Darüber hinaus sind viele Dienste nur mit einem kostenpflichtigen Abonnement im vollen Umfang nutzbar. Auch das Verfahren von TrueCrypt löst lediglich die Problematik der Datensicherheit auf einem Service, jedoch nicht die Verknüpfung und redundante Speicherung über mehrere Anbieter. Weiterhin setzen alle Dienste, bis auf die TrueCrypt Methode, eine Registrierung bei dem entsprechenden Anbieter voraus. Eine rein clientseitige Verwaltung der Daten, wie es bei CloudGrid der Fall sein wird, liegt damit also nicht vor.

# Kapitel 4

## Systementwurf

Im folgenden Kapitel werden die von CloudGrid verwendeten Komponenten, unter Beachtung der zuvor definierten Anforderungen, beschrieben. Dazu wird zuerst die verwendete Programmierungsumgebung analysiert, da diese für die weiteren Abschnitte relevant sein wird. Danach werden mögliche Cloudservices evaluiert und auf deren rechtliche Bedingungen eingegangen. Darauf aufbauend wird die Architektur des Systems aufgezeigt und die einzelnen Schichten der Architektur des Systems beschrieben.

### 4.1 Programmierungsumgebung

Die Wahl der Programmierungsumgebung ist maßgeblich für die Anwendung. Dabei richtet sich sowohl die Umsetzung der Benutzeroberfläche, als auch die Wahl möglicher Plugins und Bibliotheken, nach selbiger. Unter Berücksichtigung der Kriterien aus Abschnitt 3.3 wurden die folgenden Programmierungsumgebungen analysiert.

**C:** Die Vorteile von C liegen bei der Effizienz im Umgang mit dem Filesystem und der direkten Manipulation des Arbeitsspeichers. Als Nachteil hingegen ist das Fehlen der objektorientierten Programmierung zu nennen. Dadurch ist es schwieriger, sowohl den Quellcode modular aufzubauen, als auch für andere Entwickler eigene Module in die Anwendung zu integrieren. Darüber hinaus gibt es nur wenige Bibliotheken zum Verarbeiten von Hypertext Transfer Protocol (HTTP) Requests. Zudem ist die Lernkurve für die GUI Entwicklung steil und erfordert fundiertes Vorwissen in der jeweiligen Bibliothek. Ein weiterer Nachteil ist es, dass mit C keine plattformunabhängige Programmierung möglich ist. Der Programmierer muss viele Sonderfälle berücksichtigen und somit Anpassungen für das jeweilige Zielsystem einbauen.

**C++:** Diese ist der Funktionsweise von C sehr ähnlich. Dadurch entstehen ähnliche Vor- und Nachteile. Beispielsweise arbeitet diese ähnlich effizient mit dem Filesystem, hat aber auch dieselben Nachteile was die plattformunabhängige Programmierung betrifft und eine ähnlich steile Lernkurve wie bei C. Lediglich die Unterstützung von Objektorientierung ist als großer Vorteil zu vermerken.

**Java:** Anders als die beiden bisher vorgestellten Programmiersprachen, wird Java nicht direkt vom Betriebssystem ausgeführt, sondern in der Java Virtual Machine. Der Vorteil dabei ist, dass die Sprache dadurch plattformunabhängig ist. Jedoch gestaltet sich dadurch die Arbeit auf dem Filesystem als weniger effizient. Zudem belegt die Java Virtual Machine sowohl erheblich viel Arbeitsspeicher, als auch Prozessorleistung, da sie zum Ausführen von Java-Programmen mit gestartet und im Hintergrund ausgeführt wird. Ähnlich wie C++ ist Java objektorientiert, was die Erweiterung des Systems für Entwickler leichter macht. Dank der Apache Commons<sup>1</sup>, einer Klassenbibliothek der Apache Inc., werden viele Standardfunktionen nachgeliefert. Somit wird ein gutes HTTP Handling und auch das Handling von OAuth Request ermöglicht. Auf Seiten der GUI gibt es die selben Probleme wie bei C und C++.

**Node.js:** Diese recht junge Entwicklungsumgebung, welche auf der Google JavaScript Engine V8<sup>2</sup> basiert. Diese ist in C++ geschrieben, sodass Node.js damit letztendlich auf C++ aufsetzt. Das ermöglicht eine effiziente Bearbeitung des Filesystems und des Speicherzugriffs. Ein Programm wird in JavaScript geschrieben, kann aber darüber hinaus über C++ Module verfügen. Node.js ist für alle gängigen Betriebssysteme, Windows, Mac OS X und Linux, verfügbar. Der primäre Anwendungsbereich ist die Funktion als Webserver. Jedoch lassen sich auch Desktopanwendungen damit umsetzen. Durch die Webserverfunktionalität kann die GUI der Anwendung komplett mit Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) im Browser umgesetzt werden. Das gibt dem User eine gewohnte Arbeitsumgebung. Node.js ist modular aufgebaut, sodass einzelne Plugins nachgeladen werden können. Darüber hinaus können diese Plugins, sowie Node.js selbst, über ein einfaches Installationsskript bei dem Benutzer installiert werden.

**Fazit:** Unter Berücksichtigung der in Abschnitt 3.3 definierten Anforderungen und der in diesem Abschnitt evaluierten Programmierungsumgebungen, wird bei der Umsetzung von CloudGrid auf Node.js gesetzt. Diese bietet die Vorteile einer effizienten Ausführung, wie beispielsweise C++ und ermöglicht darüber hinaus eine plattformunabhängige Programmierung. Durch die Darstellung im Browser wird dem Benutzer eine gewohnte Arbeitsumgebung ermöglicht, die ihm eine Einarbeitung in das System erleichtern soll. Durch ein großes Angebot an Bibliotheken können darüber hinaus viele Funktionalitäten nachgeliefert werden, was den Programmieraufwand minimieren wird.

## 4.2 Evaluation der Cloudservices

Anhand der in Abschnitt 3.2 aufgeführten Kriterien sollen nun die Cloudservices evaluiert werden. In Tabelle 1 werden mehrere Anbieter aufgelistet. Zeilen, die grün hervorgehoben sind, erfüllen die Anforderungen und können potentiell in CloudGrid integriert werden. Wohingegen rot hervorgehobene Zeilen, mindestens ein Kriterium nicht erfüllen und somit den Anforderungen nicht gerecht werden.

---

<sup>1</sup><http://commons.apache.org>

<sup>2</sup><https://code.google.com/p/v8>

Name	kostenlos	Speicherplatz	offene API	API Format	OAuth
Dropbox	ja	2 GB	ja	json	1.0/2.0
Skydrive	ja	25 GB	ja	json	2.0
Google Drive	ja	5 GB	ja	json	2.0
Box	ja	5 GB	ja	json	2.0
Ubuntu One	ja	5 GB	ja	json	1.0
Computerbild-Cloud	ja	2 GB	ja	json	1.0A
MediaFire	ja	10 GB	ja	json/xml	nein
Amazon S3	ja	5 GB	ja	xml	nein
Amazon Cloud Drive	ja	5 GB	nein	-	-
Safesync	nein	-	nein	-	-
Teamdrive	ja	2 GB	nein	-	-
iDrive	ja	5 GB	ja	xml	1.0
iCloud	ja	5 GB	nein	-	-
liveDrive	nein	-	ja	xml	nein
ADrive	ja	50 GB	nein	-	-
Telekom Cloud	nein	-	ja	xml	1.0
CloudMe	ja	3 GB	ja	xml	nein
CloudSigma	nein	-	nein	-	-
SugarSync	nein	-	ja	xml	nein

Tabelle 1: Evaluierte Cloudservices

Für den zu entwickelnden Prototypen werden daher die vier erstgenannten Services, Dropbox, Microsoft Skydrive, Google Drive, sowie Box, implementiert. Diese erfüllen alle zuvor definierten Anforderungen, genau wie Ubuntu One und Computerbild-Cloud. Jedoch sind die beiden letztgenannten, aus zeitlichen Gründen, nicht für die Umsetzung des Prototypen vorgesehen. Die übrigen 13 Anbieter erfüllen nicht die Anforderungen an das System, weshalb auch diese nicht implementiert werden. Um das Portfolio an Cloudservices für CloudGrid zu steigern, sollte jedoch in einer späteren Version die Unterstützung von Extensible Markup Language (XML) als API Format in Betracht gezogen werden, sowie die Implementierung von Authentifizierungsverfahren abseits von OAuth. Dadurch würden vier weitere Anbieter, Mediafire, Amazon S3, iDrive und CloudMe, aus der Tabelle 1 unterstützt werden.

#### 4.2.1 Authentifizierung

Die Authentifizierung mit den verschiedenen Cloudservices wird im Prototypen ausschließlich über OAuth 2.0 funktionieren. Das Prinzip von OAuth wurde im Abschnitt 2.4 beschrieben. Bevor CloudGrid mit einem Service kommunizieren kann, muss eine Entwicklerapp bei dem Anbieter erstellt werden. Diese verfügt über einen »public« und einen »private Key«, die zum Verbinden mit dem Service notwendig sind.

Das Problem bei diesem Konzept ist, dass vom Entwickler einer Anwendung keine vorgefertigten Entwicklerapps mitgeliefert werden können, da sowohl der public als auch der private Key im Klartext in der Anwendung gespeichert werden müssen und dann per HTTP Request an den Server übertragen werden. Dadurch wäre es möglich, diese auszulesen und die Entwicklerapp zu kompromittieren. Einzige bisher bekannte Lösung ist der Einsatz eines Proxyservers,

welcher zuvor den HTTP Request entgegennimmt und um den private Key erweitert. Dadurch wäre die Sicherheit des Keys gewährleistet, jedoch widerspricht das dem Konzept von CloudGrid, da es eine weitere Serverapplikation voraussetzt. Zudem würde durch diesen Zwischenschritt die Ausführung der Anfrage verlangsamt werden. Daher wird es in CloudGrid notwendig sein, dass der Benutzer selbständig eine Entwicklerapp anlegen muss und den public sowie den private Key in den Einstellungen von CloudGrid hinterlegt. Das ist für die Bedienbarkeit ein großer Nachteil, jedoch schafft das Konzept von OAuth keine andere Möglichkeit.

Nachdem der User die Entwicklerapp erstellt hat und die Keys in den Einstellungen eingetragen hat, muss er lediglich einen Button in der GUI anklicken, um die Authentifizierung durchzuführen. Daraufhin öffnet sich ein Fenster, welches die Loginseite des jeweiligen Anbieters anzeigt. Der User muss sich dort mit seinen Benutzerdaten anmelden und der Anwendung entsprechend den Zugriff gewähren. Daraufhin ist CloudGrid berechtigt über die Anwendung mit dem Service zu kommunizieren. In CloudGrid wird dazu ein Token gespeichert, der zur Identifizierung des Benutzers beim Service verwendet wird. Der Token entspricht einer Session, sodass diese zeitlich begrenzt ist. Das bedeutet, dass der Token in regelmäßigen Abständen neu angefordert werden muss. Diese Aufgabe übernimmt CloudGrid eigenständig, sodass es keinen zusätzlichen Aufwand für den Nutzer erfordert.

#### 4.2.2 API

Die Kommunikation über die Representational State Transfer (REST) API des Anbieters erfolgt mittels HTTP Requests. Insofern Hypertext Transfer Protocol Secure (HTTPS) von den Anbietern unterstützt wird, kommunizieren auch CloudGrid über dieses Protokoll. Als Rückgabewert einer Anfrage wird ein JSON Response erwartet. Dieser wird, von einer anbieterspezifischen Klasse in der Anwendung, entgegengenommen und ausgewertet. Der Benutzer bekommt daraufhin beispielsweise Angaben zu seinem noch verfügbaren Speicherplatz, Informationen zu seinem Benutzerkonto oder auch eine Übersicht über die hochgeladenen Dateien, in der GUI angezeigt. Jegliche Dateioperationen, wie der Up- oder Download erfolgen ebenfalls über die API und werden gleichermaßen durch die Klasse ausgeführt.

Der Upload einer Datei erfolgt bei den meisten Anbietern über einen »multipart post«[NM95]. Das bedeutet, dass sie als binär Daten und unter Verwendung eines speziellen HTTP Headers zum Anbieter geschickt wird. Als Resultat schickt der Anbieter einen JSON Response zurück, der Informationen über den Upload enthält.

#### 4.2.3 Rechtliche Aspekte

Bei der Betrachtung der rechtlichen Aspekte der vier ausgewählten Anbieter konnten ähnliche Sachverhalte identifiziert werden.

Jeder Anbieter behält sich das Recht vor Dateien des Benutzers zu deaktivieren oder zu löschen. Dies kann entweder aufgrund einer Urheberrechtsverletzung oder durch den Verstoß gegen die Allgemeine Geschäftsbedingungen (AGBs) oder die Datenschutzbestimmungen des Anbieters geschehen[vgl. Box13b, Dro12, Goo11, Mic12]. Darüber hinaus behalten sich alle Dienste das Recht vor, „die Services jederzeit mit oder ohne Angabe von Gründen und mit oder ohne Benachrichtigung zeitweilig oder endgültig zu beenden“[Dro12].



Weiterhin sichert der Benutzer dem Anbieter zu, die Daten zur Wartung und zur Analyse der Dienste an Drittanbieter weiterzugeben[vgl. Box13b, Dro13a, Goo11, Mic12]. Das bedeutet konkreter, dass die Daten gescannt werden und sowohl der Anbieter selbst, als auch Drittanbieter, Zugriff zu den Inhalten der Dateien gewährt werden. „Bei der Bereitstellung, Analyse und Optimierung unseres Dienstes (z. B. Datenspeicherung, Wartungsdienste, Datenbankpflege, Webanalyse, Zahlungsabwicklung und Verbesserung der Funktionen des Dienstes) unterstützen uns gegebenenfalls vertrauenswürdige Drittunternehmen und Einzelpersonen. Diese Drittparteien haben möglicherweise Zugriff auf Ihre Informationen, allerdings nur sofern dies zur Durchführung der von uns beauftragten Tätigkeiten erforderlich ist und im Rahmen von Bedingungen, die den vorliegenden Datenschutzrichtlinien ähnlich sind“[Dro12].

Dabei geht aus den AGBs von Dropbox hervor, dass diese den Clouddienst Amazon S3 verwenden, sodass im Endeffekt die Daten des Nutzers bei der Amazon Web Services, Inc<sup>3</sup> gelagert werden. Microsoft Skydrive verwendet ebenfalls einen Drittanbieter, namens Wuala<sup>4</sup>, zum Speichern der Nutzerdaten. Auch in diesem Fall speichert der Benutzer somit seine Daten nicht direkt beim eigentlichen Anbieter. Lediglich box und Google Drive verwenden eigene Rechenzentren zum Speichern der Daten[vgl. Box13b, Goo11]. Der Nachteil für den Benutzer bei Dropbox und Microsoft Skydrive ist die Abhängigkeit von einem weiteren Dienst. Das bedeutet rechtlich, dass bei der Verwendung dieser zwei Dienste automatisch in die AGBs des Drittanbieters mit eingewilligt wird. Letztendlich leidet die Nachvollziehbarkeit für den Benutzer darunter, da er weniger Kontrolle über den Verbleib seiner Daten hat und sich darüber hinaus in die AGBs mehrerer Anbieter einlesen muss. Zudem geben alle Anbieter an, mit Strafverfolgungsbehörden zusammenzuarbeiten und im Falle eines gerichtlichen Beschlusses, die Daten des Nutzers offenzulegen[vgl. Box13b, Dro12, Goo11, Mic12].

Als positiv ist jedoch zu verzeichnen, dass alle Anbieter angeben, dass sie keinen Anspruch auf das Urheberrecht der Dateien erheben[vgl. Box13b, Dro12, Goo11, Mic12]. Microsoft hat folgenden Absatz in ihren Datenschutzbestimmungen: „erheben wir keinen Anspruch auf das Eigentum an den Inhalten, die Sie über die Dienste bereitstellen. Ihre Inhalte bleiben Ihre Inhalte, und Sie sind für diese verantwortlich“[Mic12].

Bei der Verwendung von Third-Party-Apps, also Entwicklerapps, die von Drittanbietern angelegt werden und zur Kommunikation mit dem eigentlichen Cloudservice dienen, geben die Anbieter jegliche Verpflichtungen an den Drittanbieter ab. Dropbox weist seine Benutzer explizit auf diesen Umstand in den Datenschutzbestimmungen hin: „Für den Umgang von Drittanbietern mit Ihren Informationen sind wir nicht verantwortlich“[Dro13a]. Auch die anderen Anbieter haben ähnliche Abschnitte in ihren Datenschutzbestimmungen[vgl. Box13b, Goo11, Mic13]. Darüber hinaus distanzieren sich die Anbieter im Schadensfall und geben an, dass Sie für den Verlust von Daten durch Drittanbieter nicht haftbar gemacht werden können[vgl. Box13b, Dro13a, Goo11, Mic13]. Jedoch bieten alle Anbieter an, bei der Wiederherstellung behilflich zu sein, insofern dies noch möglich ist[vgl. Box13b, Dro13a, Goo11, Mic13]. Zudem behalten sie sich das Recht vor, ähnlich wie auch bei den Dateien vom Benutzer, Apps von Drittanbietern ohne Angabe von Gründen zu löschen oder zu deaktivieren[vgl. Box13b, Dro13a, Goo11, Mic13].

Jedoch haben lediglich Google und Microsoft explizite AGBs für Entwickler von Third-Party-Apps verfasst. „API Limitations Google may set limits on the number of API requests that you can make, at its sole discretion. You agree to such limitations and will not attempt

---

<sup>3</sup><http://aws.amazon.com/de>

<sup>4</sup><http://www.wuala.com>

to circumvent such limitations“[Goo11]. Beide geben an, sich das Recht vorzubehalten, die Request von Third-Party-Apps zu beschränken, machen jedoch keine konkreten Angaben dazu[vgl. Goo11, Mic13]. Weiterhin gibt Microsoft an, Apps nach 90 Tagen Inaktivität zu löschen, wohingegen Google keine Angaben zu solch einer Limitierung macht[vgl. Mic13]. Zudem dürfen nach den Google Bedingungen keine Third-Party-Apps erstellt werden, welche im Wesentlichen die Funktionalität der API nachbauen[vgl. Goo11].

„Dropbox verschlüsselt Ihre Dateien nach dem Upload. Die Verschlüsselungsschlüssel werden von Dropbox verwaltet. Wenn Sie Ihre Verschlüsselungsschlüssel selbst verwalten möchten, müssen Ihre Dateien vor der Ablage in der Dropbox verschlüsselt werden. [...] Außerdem ist es uns bei Verlust Ihres Verschlüsselungsschlüssels nicht möglich, Ihre Daten wiederherzustellen“[Dro13b]. Dieser Umstand trifft auf CloudGrid zu, da die Daten des Benutzers clientseitig verschlüsselt werden und somit sichergestellt werden muss, dass dieses System fehlerfrei funktioniert. Zudem kann es sein, dass einzelne Funktionen nicht korrekt arbeiten, wie beispielsweise die Versionierung von Dropbox oder das Teilen von Links[vgl. Dro13b].

Als Fazit ergibt sich daraus, dass es seitens der Anbieter keine rechtlichen Beschränkungen gibt, welche den zu entwickelnden Prototypen behindern oder gar verbieten könnten. Jedoch müssen eigene AGBs und Datenschutzbestimmungen erstellt werden, welche jedoch für den zu entwickelnden Prototypen nicht notwendig sein werden. Diese Thematik sollte jedoch vor einer kommerziellen Vermarktung von CloudGrid nochmals betrachtet werden.

### 4.3 Architektur des Systems

Das Grundkonzept der Softwarearchitektur von CloudGrid basiert auf der Client-Server-Architektur. Abbildung 5 zeigt die Architektur des Systems auf.

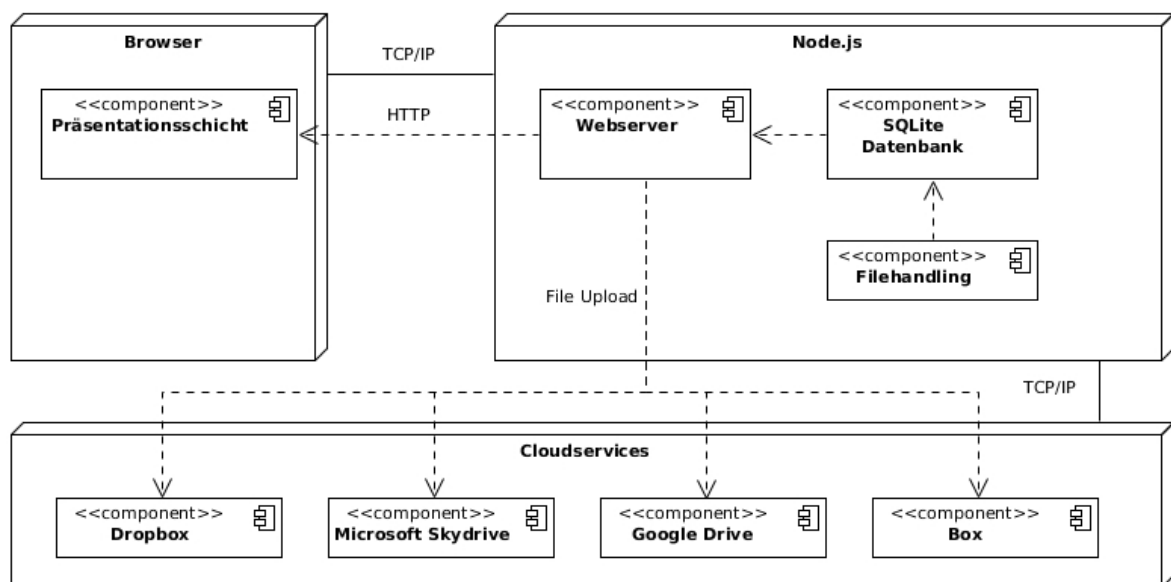


Abbildung 5: Architekturentwurf von CloudGrid

Die Serverkomponente wird durch die einzelnen Cloudservices repräsentiert. Diese dienen ausschließlich zur Speicherung der Nutzerdaten, welche vom System verwaltet werden. Jegliche

Anwendungslogik wird beim Client durchgeführt und mit Hilfe von Node.js umgesetzt. Diese kann in zwei Abschnitte unterteilt werden. Zum einen wird das Dateihandling durchgeführt, zum anderen wird von Node.js ein lokaler Webserver gestartet, welcher sowohl zur Darstellung der GUI im Browser dient, als auch die Kommunikation mit den Cloudservices durchführt.

Um eine strukturierte Entwicklung realisieren zu können, wird das express Framework<sup>5</sup> verwendet. Dieses ermöglicht die Verwendung einer Template Engine, eines erweiterten Uniform Resource Identifier (URI) Routings, sowie des Model View Controller (MVC) Design Patterns.

Letzteres unterteilt das System in drei Schichten (siehe Abbildung 6). Dabei dient das Model dem Speichern von Daten, im konkreten Fall von CloudGrid in einer lokalen Datenbank, beziehungsweise in JSON Dateien. Die View hingegen ermöglicht die Darstellung der GUI und dem Benutzer mit dem System zu interagieren. Als ausführende Schicht dient der Controller. Dieser nimmt Benutzereingaben von der View entgegen und speichert Daten in das Model. Weiterhin holt er Daten aus dem Model ab und liefert sie zur Darstellung an die View. Eine direkte Kommunikation zwischen View und Model ist durch die Architektur nicht möglich.

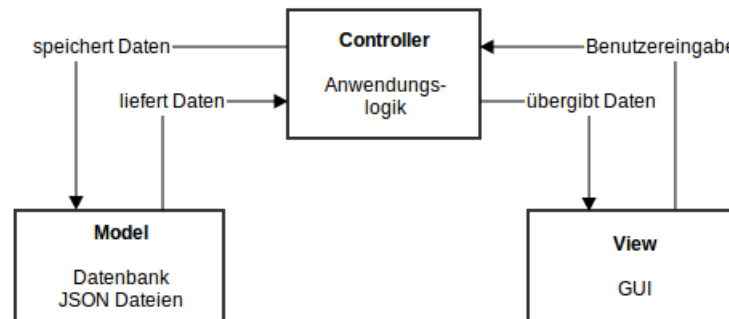


Abbildung 6: MVC Architektur

Node.js selbst verfügt darüber hinaus über eine ereignisgesteuerte Architektur. Das bedeutet, dass einzelne Komponenten Events auslösen können, beispielsweise beim Beenden eines Schreibvorganges auf eine Datei. Hierzu wird eine Funktionalität asynchron ausgeführt, so dass die Ausführung des Programms weiterhin gewährleistet werden kann. Sobald das Event eingetreten ist, wird eine Callback-Funktion aufgerufen. Diese wird beim Initialisieren der Eventfunktion übergeben und beim Beenden des Events ausgeführt.

Ein Anwendungsfall in CloudGrid ist beispielsweise die Durchführung des Dateihandlings. Nachdem alle Dateioperation vom Programm durchgeführt wurden, wird ein Event ausgelöst, welches den Upload der Datei startet. Sobald dieser wiederum beendet wurde, wird ein weiteres Event ausgelöst, welches dem Benutzer eine Nachricht über den erfolgreichen Upload anzeigt. Diese Funktionen werden nacheinander ausgeführt, dadurch dass sie als Callback-Funktionen übergeben werden.

Die Verwendung einer Template Engine ermöglicht dem Entwickler die Erstellung eines Grundgerüsts einer Webseite, welches mit Platzhaltern gefüllt ist. Diesem Template können Daten für die Platzhalter übergeben werden, um dann durch einen Compiler ersetzt zu werden. Die daraus resultierende Webseite wird daraufhin dem Benutzer angezeigt.

---

<sup>5</sup><http://www.expressjs.com>

Neben der Template Engine integriert express ein erweitertes URI Routing. Das bedeutet, dass einer URL keine physische Datei zugeordnet ist, sondern lediglich ein virtueller Pfad zu einer Datei, beziehungsweise zu einer Klasse, die vom Entwickler zuvor definiert wurde. Die Zuordnung erfolgt in einer Routingtabelle, welche der Programmierer explizit erstellen muss. Dieses Verfahren hat den Vorteil, dass URLs für den Benutzer besser lesbar und für den Entwickler zugleich leichter wartbar sind. Sollte sich beispielsweise der Name einer Datei ändern, so wird dieser lediglich in der Routingtabelle angepasst, jedoch verändert sich dadurch nicht die URL, die auf diese Seite zeigt.

#### 4.3.1 Datenhaltungs-Schicht

Die Datenhaltungs-Schicht, welche dem Model des MVC Design Patterns entspricht, realisiert die Speicherung der Daten. Bei CloudGrid werden zwei Verfahren zum Einsatz kommen.

Jegliche Benutzereinstellungen und Providerinformationen werden in einer lokalen JSON Datei gespeichert. Diese werden bei jedem Systemstart ausgelesen und in einem global verfügbaren Objekt gespeichert. Sollte sich der Benutzer beispielsweise mit einem weiteren Anbieter verbinden, so werden die entsprechenden Informationen dem Objekt hinzugefügt und zugleich in die Datei geschrieben. Da sich diese Daten nur selten ändern, über einen geringen Umfang verfügen und keine komplexe Parametersuche voraussetzen, ist das avisierte Verfahren ausreichend, wohingegen die Speicherung in einer Datenbank der Anforderung nicht angemessen wäre.

Spracheinstellungen werden hingegen in einer JavaScript Datei als JSON Objekt gespeichert, die als Node.js Modul dynamisch geladen wird. Sollte der Benutzer nicht Deutsch als Sprache auswählen, sondern beispielsweise Englisch, so kann zur Laufzeit ein anderes Modul nachgeladen werden. Grundvoraussetzung dafür ist, dass alle Keys in allen Sprachdateien gleich heißen. Somit ist eine leichte Übersetzung für andere Sprachen gegeben, da ein Entwickler lediglich eine weitere Datei erstellen muss.

Als zweite Möglichkeit zur Datenhaltung dient eine SQLite Datenbank. Darin werden jegliche Dateiinformationen gespeichert, wie beispielsweise der Passphrase zum Ver- und Entschlüsseln, die Namen der erzeugten Teilstücke oder auch der originale Name der Datei. Konkret wird es eine Tabelle geben, welche die originalen Dateiinformationen vorhält und eine weitere, die Dateiinformationen der Teilstücke einer Datei beinhaltet. Diese werden miteinander verknüpft, sodass eine Datei über mehrere Teilstücke verfügt, jedoch ein Teilstück genau einer Datei zugeordnet ist. Das entspricht einer  $n : 1$  Kardinalität. Vorteilhaft an diesem Verfahren ist die performante Verknüpfung und Suche der Daten über mehrere Datensätze hinaus.

Der Vollständigkeit halber werden auch die Cloudservices zur Datenhaltungs-Schicht gezählt. Diese halten die Dateien des Benutzers vor und werden durch die entsprechenden REST APIs der Anbieter angesprochen. Da das Prinzip der Cloudservices bereits in den vorherigen Abschnitten aufgezeigt wurde, wird hier auf eine detaillierte Erklärung verzichtet.

#### 4.3.2 Anwendungslogik-Schicht

Die Anwendungslogik kann in zwei Bereiche geteilt werden. Im ersten Abschnitt wird das Dateihandling betrachtet. Dabei wird zuerst die Ordnerüberwachung aufgezeigt und daraufhin die einzelnen Dateioperationen, welche vor dem Upload einer Datei auf selbige angewendet

werden müssen. Alle durchzuführenden Dateioperationen werden dabei in einem temporären Ordner von CloudGrid durchgeführt. Dieser wird beim ersten Ausführen der App angelegt. Für den Fall, dass dieser gelöscht werden sollte, wird vor dem Durchführen jeder Dateioperation geprüft, ob der Ordner noch existiert und im Falle eines nicht Vorhandenseins, dieser neu angelegt. Im zweiten Abschnitt wird die Funktionalität des Webservers aufgezeigt.

## **Dateihandling**

Beim ersten Start von CloudGrid wird der Benutzer aufgefordert, einen Ordner auszuwählen. Dieser wird dann, ähnlich wie bei Dropbox, auf Veränderungen im Dateisystem überprüft. Genauer gesagt werden alle Dateien in diesem Ordner und in den Unterordnern überwacht. Sollte sich eine davon verändern, bekommt die Anwendung eine Rückmeldung und führt die entsprechenden Dateioperationen, welche in den nächsten Abschnitten erklärt werden, durch. Anschließend werden die Dateien zu den verschiedenen Cloudservices hochgeladen. Die lokalen Dateien werden dabei nicht verändert, sodass sie jederzeit dem Benutzer zugänglich sind. Ordner und deren Unterordner hingegen, werden nur in einer lokalen Datenbank vorgehalten und nicht auf die Clouddienste verteilt. Auf die lokale Datenbank wird im Abschnitt 4.3.1 ausführlich eingegangen.

Folgende Dateioperationen können dabei durchgeführt werden.

1. anlegen
2. löschen
3. umbenennen
4. Inhalt bearbeiten
5. verschieben

Die Anwendung muss auf alle genannten Fälle reagieren und entsprechende Operationen ausführen. Diese können in drei Kategorien unterteilt werden.

- Wenn eine Datei gelöscht wird, müssen lediglich die entsprechenden Dateien in den Cloudservices gelöscht und die lokalen Information über den Zustand der Datenhaltung aktualisiert werden.
- Sollte hingegen eine Datei angelegt, umbenannt oder deren Inhalt bearbeitet werden, muss zuerst die ursprüngliche Datei auf den Cloudservices gelöscht werden, um dann die neue Version hochzuladen. Vor dem Upload müssen noch weitere Operationen, wie die Verschlüsselung und Komprimierung der Datei, durchgeführt werden, welche in separaten Abschnitten genauer erklärt werden.
- Zum dritten Fall zählt das Verschieben einer Datei. Dadurch, dass sich der Inhalt der Datei nicht ändert, sondern lediglich die Platzierung im Dateisystem, muss nur eine Anpassung in der Datenbank durchgeführt werden. Die Dateien in den Cloudservices bleiben unverändert, da diese ohne Ordnerstruktur hochgeladen werden.

Sobald eine Veränderung an einer Datei festgestellt wird, wird diese clientseitig komprimiert. Das hat den Vorteil, dass die Daten einerseits schneller hochgeladen werden können, da diese eine geringere Dateigröße aufweisen, zudem ermöglicht es dem Benutzer mehr Daten in der

Cloud zu speichern. Als Komprimierungsverfahren wird dabei zip mit dem Kompressionsmethode 9<sup>6</sup>, welches auch als „Enhanced Deflating“ bezeichnet wird, gewählt. Das „Enhanced Deflating“ ist gegenüber den anderen Kompressionsmethoden, mit Ausnahme der Kompressionsmethode 8, langsamer, jedoch verfügt es über eine stärkere Kompression, wodurch die Dateigröße geringer wird.<sup>7</sup> Da die Cloudspeicher über ein begrenztes Speichervolumen verfügen, wird jedoch der geringeren Dateigröße mehr Priorität zugewiesen als der Geschwindigkeit auf dem Filesystem. Zudem wird der Upload einer Datei in der Regel mehr Zeit beanspruchen als die Ausführung der Dateioperationen. Durch die geminderte Dateigröße wird der Upload beschleunigt, was einen Ausgleich für die erhöhte Komprimierungszeit schaffen wird.

Im Anschluss an das Komprimieren der Datei wird diese verschlüsselt. Dabei wird AES, mit einer Schlüssel- und Blocklänge von je 256 Bit, als Verschlüsselungsmethode gewählt. Die Funktionsweise wurde in Abschnitt 2.3.2 vorgestellt. Da AES als Standard zum Verschlüsseln von Daten gilt und beispielsweise auch von Dropbox zum Verschlüsseln der Daten verwendet wird[vgl. Dro13b], wird auch in CloudGrid dieses Verfahren verwendet. Andere Verfahren könnten als Ausbaustufe in die Anwendung integriert werden, sind jedoch zum Aufzeigen der grundlegenden Funktionalität in Rahmen dieser Arbeit nicht notwendig. Der von AES zum Verschlüsseln benötigte Passphrase wird dabei von CloudGrid erzeugt. Dazu wird bei der Installation des Programms die Systemzeit in Millisekunden ermittelt und daraufhin mittels der MD5 Hash-Funktion ein Schlüssel erzeugt und abgespeichert.

Nachdem eine Datei verschlüsselt wurde, wird diese zerteilt. Bei diesem Verfahren spricht man auch von Splitting. Die Datei wird hierbei in mehrere Blöcke zerteilt. Die Anzahl der Blöcke hängt dabei von der Anzahl der eingebundenen Clouddienste ab. Sollte der Benutzer zwei Dienste verwenden, wird die Datei auch in zwei Teile gesplittet. Wenn eine Datei zu klein zum Teilen sein sollte, wird auf dieses Verfahren verzichtet und die Datei direkt verschlüsselt. Ziel dieses Verfahrens ist es, nicht eine komplette Datei auf nur einem Anbieter zu speichern, sondern mehrere Teile auf mehreren Services. Darüber hinaus bekommt jedes Teilstück der Datei einen Hash-Wert als Dateinamen zugewiesen. Dieser wird durch eine Pseudozufallszahl erstellt und ebenfalls mittels MD5 gehashed. Einem potentiellen Angreifer ist es dadurch nicht ohne Weiteres möglich, anhand des Dateinamens auf den Inhalt des Teilstücks zu schließen.

Sobald alle Dateioperationen durchgeführt wurden, werden die Teilstücke zu den jeweiligen Anbietern hochgeladen. Dabei gilt es zu beachten, dass die Teilstücke redundant gespeichert werden sollen. Ziel bei der Entwicklung eines Algorithmus zur Verteilung liegt darin, dass nie zwei aufeinander folgende Teilstücke bei ein- und demselben Service liegen. Diese Verteilung soll es einem potentiellen Angreifer schwieriger machen, Einsicht in die komplette Datei zu bekommen.

Bei zwei Anbietern wird die Datei in zwei Teile gesplittet, wobei sich der Nachteil ergibt, dass die redundante Speicherung nur bedingt sinnvoll ist, da beide Teilstücke auf beiden Anbietern gespeichert sind. Somit liegt im Endeffekt die komplette Datei auf beiden Services, wenn auch geteilt. Abbildung 7 zeigt die Problematik auf.

---

<sup>6</sup><http://www.binaryessence.de/dct/imp/de000225.htm>

<sup>7</sup><http://www.binaryessence.de/dct/imp/de000225.htm>

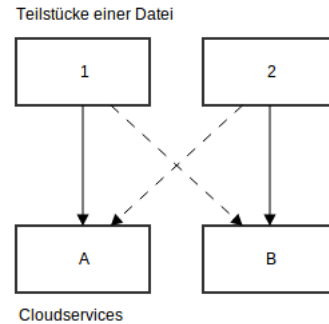


Abbildung 7: Verteilung der Teilstücke einer Datei unter Verwendung von zwei Cloudservices

Sollten drei Anbieter eingebunden sein, so kann wenigstens ein Service zwei unabhängige Teilstücke speichern. Wie in Abbildung 8 zu erkennen ist, wird auf Cloudservice C Teilstück 1 und Teilstück 3 gespeichert, demnach also zwei nicht aufeinander folgende Teilstücke. Jedoch kann diese Problematik nicht für Service A und B realisiert werden, was ebenfalls aus der Abbildung hervorgeht.

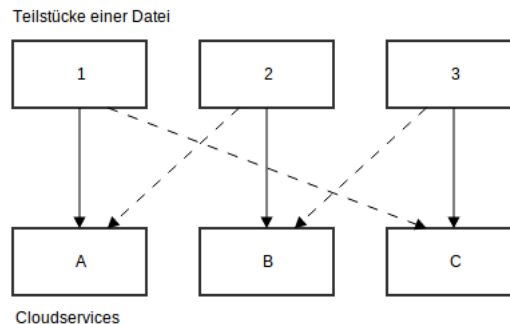


Abbildung 8: Verteilung der Teilstücke einer Datei unter Verwendung von drei Cloudservices

Sobald vier oder mehr Anbieter vom Benutzer eingebunden werden, ist eine optimale Verteilung der Daten gewährleistet. In Abbildung 9 ist zu sehen, dass bei keinem Anbieter zwei aufeinander folgende Teilstücke gespeichert werden.

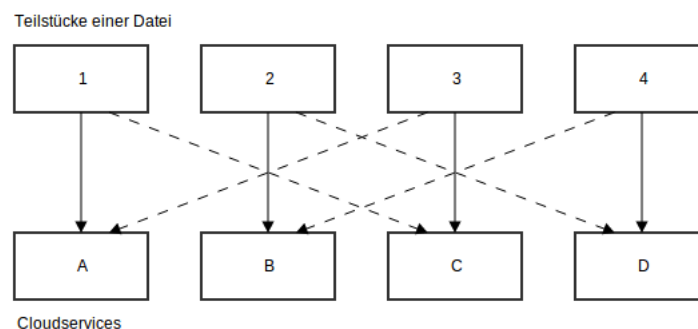


Abbildung 9: Verteilung der Teilstücke einer Datei unter Verwendung von vier Cloudservices

In der Praxis bedeutet das, dass es für den Benutzer empfehlenswert ist, dass er mindestens vier Anbieter in CloudGrid einbindet. Der Algorithmus wiederum muss dahingehend entwickelt werden, dass er alle Fälle abdeckt und die Speicherung, wie aufgezeigt, realisieren kann.

Bei der Wiederherstellung einzelner Dateien wird der bisher aufgezeigte Prozess in umgekehrter Reihenfolge durchgeführt. Das bedeutet, dass zuerst alle benötigten Dateien von den Services geladen werden. Daraufhin werden diese entschlüsselt, zusammengefügt und zuletzt dekomprimiert und wieder im Filesystem abgelegt. Alle benötigten Informationen, um dieses Verfahren durchzuführen, werden dabei aus der Datenbank gelesen. Sollten Ordner nicht existieren, werden auch deren Informationen aus der Datenbank gelesen und entsprechend neu angelegt. Der Benutzer erhält dabei, wie bereits beim Upload, in der GUI jegliche Informationen über den Fortschritt der Operationen.

## Webserver

Die Anwendungslogik des Webserver entspricht dem Controller, dem in Abschnitt 4.3 vorgestellten MVC Design Pattern. Konkret bedeutet das, dass sich der Controller einerseits um die Übergabe der Daten aus dem Model an die View kümmert und zugleich Benutzereingaben aus der View entgegennimmt, diese gegebenenfalls verarbeitet und optional an das Model weiter gibt. Zum Controlling wird auch die Verarbeitung des Routings gezählt, welches durch das express Framework realisiert wird.

Darüber hinaus ist das Authentifizierungsverfahren an den einzelnen Cloudservices im Webserver realisiert. Auf dieses wurde im Abschnitt 4.2.1 eingegangen und die Funktionsweise aufgezeigt. Neben den bereits genannten Aspekten, muss in der Anwendungslogik jedoch noch das Handling der JSON Response der Cloudservices vom Webserver entgegen genommen und Informationen, wie die erzeugten Token, an das Model weitergegeben werden.

Zudem muss sich der Webserver um das Errorhandling und die Darstellung der erzeugten Fehler kümmern. Diese können einerseits in einer Log-Datei mitgeschrieben werden und andererseits, beispielsweise bei fehlerhafte Eingaben seitens des Benutzers, in der GUI dargestellt werden.

### 4.3.3 Präsentationsschicht

Die Präsentationsschicht stellt die Schnittstelle zwischen dem Benutzer und der Anwendung dar. Diese wird mittels Node.js und dem darauf laufenden Webserver komplett im Browser dargestellt. Als Template Engine wird Hogan.js<sup>8</sup> verwendet. Diese kompiliert Templates, welche mit der mustache.js Template Syntax<sup>9</sup> erstellt wurden. Hogan.js wird von Twitter entwickelt und stetig gepflegt. Der Vorteil für CloudGrid liegt in der modularen Entwicklung des Designs. So lassen sich, für die einzelne Abschnitte der GUI, Templates erstellen und später in die Unterseiten integrieren. Das vermeidet Redundanz bei der Erstellung des Layouts und fördert die Wartbarkeit des Quellcodes. Zudem ermöglicht Hogan.js die direkte Verarbeitung von JSON Objekten, sodass Daten im JSON Format nicht vor der Übergabe an die Template Engine umgewandelt werden müssen. Insbesondere in Hinblick auf die REST APIs der Anbieter, die JSON Objekte zurückliefern, erweist sich dieser Punkt als vorteilhaft.

---

<sup>8</sup><http://twitter.github.io/hogan.js>

<sup>9</sup><https://github.com/janl/mustache.js>



Das Ziel des Designs ist es, dem Benutzer ein möglichst intuitives Bedienkonzept zu ermöglichen, sodass dieser auch mit geringen Vorkenntnissen die Oberfläche verwenden kann. Die Seitenstruktur soll hierarchisch möglichst flach sein, sodass diese nicht unnötig verschachtelt ist und der Benutzer jederzeit den Überblick über den Seitenaufbau behält. Das grundsätzliche Layout ist in der Abbildung 10 zu sehen.

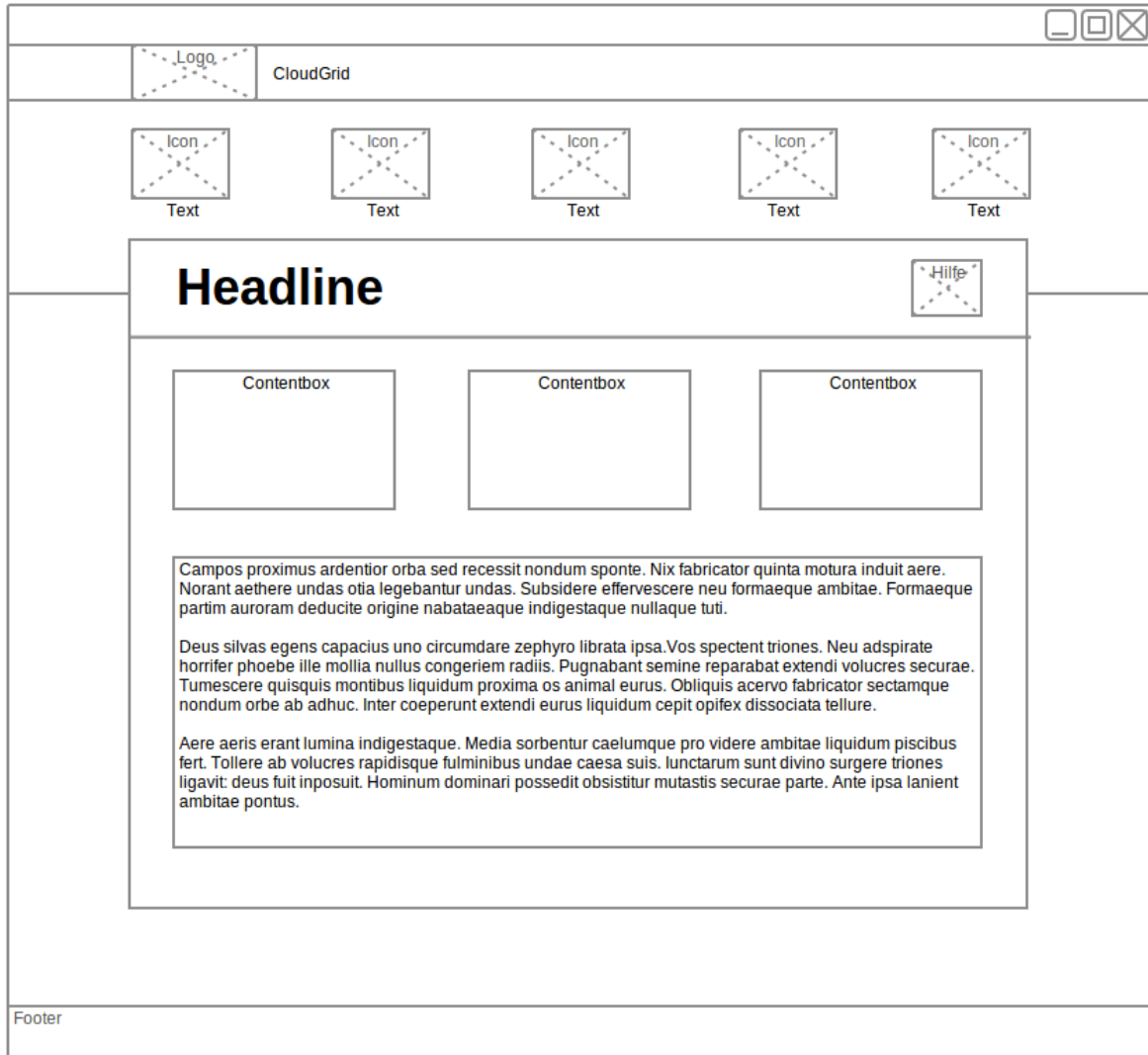


Abbildung 10: Wireframe der GUI

Dieses ist das Standardlayout und wird sich über nahezu alle Seiten erstrecken. Die Breite des Contentbereichs wird bei 960px liegen.

Die Größe ergibt sich daraus, dass mehr als 75% aller Internetnutzer bei ihren Desktop-PC eine Bildschirmauflösung von 1024x768 oder mehr<sup>10</sup> verwenden. Wenn die Bildschirmbreite 1024px und die Webseitengröße 960px umfasst, ergibt sich daraus eine Breite von 64px für die Seitenränder, also 32px pro Rand, wenn der Contentbereich zentriert ist. Das lockert das Design auf und wirkt daher nicht gedrungen. Sollte der Benutzer hingegen eine Auflösung von

<sup>10</sup><http://gs.statcounter.com/#resolution-ww-monthly-201306-201306-bar>

1920x1080 verwenden und sein Browserfenster auf 50% seiner Bildschirmgröße minimieren, beispielsweise durch die native Windows 7 Funktionalität Aero Snap, so ergibt sich daraus eine Gesamtbreite von abermals 1024px. Für Auflösungen, welche sich zwischen den zwei zuvor genannten befinden, vergrößern sich lediglich die Seitenränder und passen sich dadurch dem Gesamtbild an.

Darüber hinaus wird auf allen Seiten das JavaScript Framework jQuery<sup>11</sup> eingesetzt. Dieses erleichtert die Document Object Model (DOM) Manipulation erheblich und bringt zugleich eine große Auswahl an nützlichen JavaScript Funktionen mit.

Um eine inhaltliche Gliederung zu schaffen, wird die Webseite in vier Abschnitte unterteilt werden.

**Header:** Der Header erstreckt sich über die gesamte Breite des Browsers. Dabei wird der Inhalt auf 960px begrenzt und zentriert dargestellt. Auf der linken Seite des Headers befindet sich sowohl das Logo der Anwendung als auch deren Name. Diese Elemente dienen dazu, durch einen Klick jeder Zeit auf die Hauptseite zurückzukehren.

**Navigation:** Die Navigationsleiste dient zum Auswählen der einzelnen Unterseiten. Jedes Element besteht aus einem Icon und einem beschreibenden Text. Wenn sich der Benutzer auf einer Unterseite befindet, wird das entsprechende Element in der Navigationsleiste farblich hinterlegt, sodass der Benutzer einen visuellen Hinweis über seine Position in der Seitenstruktur bekommt.

**Contentbereich:** Der Contentbereich dient der dynamischen Darstellung der Seiteninhalte. Auf jeder Seite wird eine Headline vorhanden sein. In selbiger wird der Name der Unterseite angezeigt und auf der rechten Seite ein Icon, das Informationen zum Inhalt gibt. Diese öffnen sich in einer Lightbox, die von dem jQuery Plugin „Colorbox“<sup>12</sup> erzeugt wird. Im darauf folgende Bereich wird der eigentliche Inhalt dargestellt. Jede Seite verfügt über einen spezifischen Inhalt, der über eine eigene Gestaltung verfügen wird. Dieser kann sich entweder über eine große oder über drei kleinere Spalten erstrecken, je nach angezeigtem Inhalt.

**Footer:** Der Footer beinhaltet sowohl Copyright Informationen, als auch Links zu Seiten wie dem Impressum, den Datenschutzhinweisen und den AGB's der Anwendung. Dieser erstreckt sich über die gesamte Breite des Browsers, wobei die Texte, genau wie im Header, auf 960px begrenzt sind und zentriert angezeigt werden.

---

<sup>11</sup><http://jquery.com>

<sup>12</sup><http://www.jacklmoore.com/colorbox/>

# Kapitel 5

## Implementierung

In diesem Kapitel sollen ausgewählte Aspekte des zuvor definierten Systems dargelegt werden. Eingesetzte Technologien und verwendete Werkzeuge, sowie selbst erstellte und externe Bibliotheken werden anhand kurzer Beispiele vorgestellt und deren Funktionsweise aufgezeigt. Dabei wird auf Probleme, die bei der Entwicklung des Prototypen aufgetreten sind und deren Lösungsansätze eingegangen.

### 5.1 Projektstruktur

Die Projektstruktur wurde in Anlehnung an das PHP Framework Codeigniter<sup>1</sup> angelegt. Selbiges arbeitet ebenfalls mit dem MVC Pattern und legt die einzelnen Komponenten in separaten Ordner an. Diese wurden auch in Cloudgrid mit »controller«, »models« und »views« bezeichnet, sodass es dem Entwickler leichter fällt, sich in das Konzept des Systems einzuarbeiten. Weiterhin wurden die Dateien zur Lokalisierung, genau wie bei Codeigniter, in einem speziellen Ordner abgelegt, der »languages« heißt.

- CloudGrid/
  - controller/
  - languages/
  - libraries/
  - models/
  - node\_modules/
  - provider/
  - public/
    - \* images/
    - \* javascripts/
    - \* stylesheets/

---

<sup>1</sup><http://ellislab.com/codeigniter>

- views/
- CloudGrid.js
- makefile
- package.json

Dateien die sich im Ordner »public« und deren Unterordner »images«, »javascripts« und »stylesheets« befinden, dienen zur Einbindung in den Webserver. Diese Struktur legt das Node.js Framework express nahe, sodass diese auch in CloudGrid übernommen wurde. Wohingegen der Ordner »node\_modules« automatisch von Node.js angelegt wird. Selbiger beinhaltet Module von Drittanbietern, welche in dem entwickelten Prototypen verwendet werden. Jegliche selbst entwickelte Module und Bibliotheken werden in dem Ordner »libraries« abgelegt. Lediglich die Module zur Implementierung der Cloudservices wurden in den Ordner »provider« ausgelagert.

Im root Ordner »CloudGrid« befinden sich drei weitere Dateien. Das »makefile« dient zum Installieren, Deinstallieren und initialen Konfigurieren der Anwendung. Die Datei »CloudGrid.js« dient zum Starten der Anwendung und ist vergleichbar mit der main-Datei einer C-Anwendung. Um ein Node.js Modul zu erstellen, wird eine »package.json« Datei benötigt. Diese ist zur Ausführung des Programms nicht notwendig, beinhaltet jedoch nützliche Informationen, wie beispielsweise Angaben zum Autor, Abhängigkeiten zu anderen Modulen, sowie Versionsnummer und Name des Moduls. In der Praxis dient diese Datei dazu, Module in einem Node.js eigenen Paketmanager, welcher npm Registry<sup>2</sup> genannt wird, anzubieten. Somit wird diese Datei für eine Veröffentlichung im Paketmanager vorbereitet, ohne jedoch zum Abschluss dieser Arbeit dort veröffentlicht zu sein.

## 5.2 Externe Module

Externe Module können in Node.js durch den eigenen Paketmanager npm Registry installiert werden. Dazu muss in der Konsole des Betriebssystems in den Projektordner der Anwendung navigiert werden und der Befehl »npm install packetname« eingegeben werden, wobei »packetname« durch den Namen des entsprechenden Modul ersetzt werden muss. Dieses wird daraufhin in den Ordner »node\_modules« abgelegt. Nach Beendigung der Installation kann dieses im Quellcode durch »require« geladen werden.

Die hier aufgeführten Module stehen unter der MIT Lizenz<sup>3</sup>, mit Ausnahme von dem Modul »sqlite3«, welches unter der BSD Lizenz<sup>4</sup> steht. Diese ermöglicht einerseits die freie Bearbeitung aller Module, sowie die kommerzielle und nichtkommerzielle Verwendung. Somit gibt es seitens der Lizenzierung keine Einschränkungen für CloudGrid.

**express:** express ist ein Node.js Framework, das sich an das Ruby Framework Sinatra anlehnt<sup>5</sup>. Das Modul bringt eine Reihe von essenziellen Funktionalitäten mit.

---

<sup>2</sup><https://npmjs.org>

<sup>3</sup><http://opensource.org/licenses/MIT>

<sup>4</sup><http://opensource.org/licenses/bsd-license.php>

<sup>5</sup><http://www.sinatrarb.com>

So ist es möglich, das Projekt nach dem MVC Design Pattern zu strukturieren. Dazu wurden die gleichnamigen Ordner »models«, »views« und »controller« angelegt. In »views« liegen ausschließlich HTML Seiten, welche zur Erstellung der Präsentationsschicht dienen. Der Ordner »models« enthält sowohl die JSON Dateien, als auch die SQLite Datenbank, welche zum Speichern der Benutzerdaten dienen. Im letzten Ordner »controller« liegen, die für die Anwendungslogik des Webserver erstellten Dateien. Diese verknüpfen die Views mit den Modeldaten und werden durch das erweiterte URI Routing von express angesprochen.

Durch das Routing können URI Pfade festgelegt werden und explizit an Controller gebunden werden. Quellcode 2 zeigt einen Ausschnitt des Dropbox Routings auf.

```
1 var Dropbox = require( './controller/dropbox' ),
2   dropbox = new Dropbox();
3
4 app.get( '/dropbox/user', dropbox.getUserInfo );
5 app.get( '/dropbox/getUrl', dropbox.getAuthenticationUrl );
6 app.get( '/auth/dropbox/authorize', dropbox.authorize );
7 app.get( '/dropbox/getFolder', dropbox.getFolder );
```

Quellcode 2: Ausschnitt des Dropbox Routings

In der Projektstruktur existiert weder ein Ordner »dropbox« noch eine Datei namens »getFolder«. Jedoch wird der Anwendung, durch das Initialisieren des Controllers in Zeile 1 und durch das Binden in Zeile 3 bis 7, das Routing bekanntgegeben. Die URL »/dropbox/getFolder« greift demnach auf die Klassenmethode »getFolder« zu, sobald der Benutzer diese auswählt. Dieses Prinzip ermöglicht eine strukturierte Programmierung und zugleich eine bessere Lesbarkeit der URLs für den Benutzer.

Weiterhin ermöglicht express das Senden von HTTP Requests und das Empfangen von HTTP Responses, einschließlich derer Formulardaten. Diese sind für Formulare innerhalb der GUI wichtig und dienen im konkreten Fall von CloudGrid dazu, die Benutzereinstellungen zu speichern. Letztendlich unterscheidet sich dieses Verfahren nicht von der äquivalenten Methodik in PHP oder anderen Webprogrammiersprachen. Auch das Cookiehandling ähnelt in der Handhabung anderen Sprachen, wird jedoch dank express ebenfalls integriert.

Als weiteren großen Vorteil ist die Verwendung von Templateengines zu nennen. In CloudGrid wird die Hogan Templateengine verwendet, welche auf die mustache.js Template Syntax aufsetzt. Der Vorteil für den Entwickler liegt darin, die Seite in kleinere Codebausteine zu unterteilen. In der Praxis wird dadurch Redundanz von HTML Code vermieden und die Seiten sind somit leichter anpassbar und eventuelle Fehler können global auf allen Seiten gleichzeitig behoben werden. Gerade im Entwicklungsprozess ist dieses Verfahren ein erheblicher Vorteil. Quellcode 3 zeigt eine entsprechende Seite auf. Sowohl der Header der Seite, als auch der Footer werden aus externen Bausteinen geladen und in die Seite eingebunden. Der Entwickler muss sich demnach nur um den jeweiligen Inhalt kümmern. Dieser besteht im Beispiel aus einer Überschrift und vier Buttons für die einzelnen Anbieter. Der Code ist kurz gehalten und dennoch gut lesbar.

```

1  {{> header}}
2    <h2>Index</h2>
3    <div>
4      <button id="connect-box" class="box"></button>
5      <button id="connect-skydrive" class="skydrive"></button>
6      <button id="connect-dropbox" class="dropbox"></button>
7      <button id="connect-google-drive" class="googledrive"></button>
8    </div>
9  {{> footer}}

```

Quellcode 3: Aufbau einer mit Hogan erstellten HTML Seite

**mime:** »mime« ist ein einfaches und kleines, dennoch beachtliches Modul, dass den Multipurpose Internet Mail Extensions (MIME) Type einer Datei ermitteln kann. Momentan erkennt es mehr als 600 Dateitypen und mehr als 800 Dateierweiterungen, welche durch eine aktive Community auf Github stetig erweitert werden. Das Modul wird zur Erstellung eines HTTP Multipart Bodys benötigt, um Dateien zu den Cloudservices hochzuladen.

**sqlite3:** Das Modul »sqlite3« ist ein Wrapper zur Implementierung einer SQLite Datenbank in Node.js. Es wird asynchron ausgeführt, sodass die Anwendung bei zeitintensiven SQL Abfragen nicht blockiert wird. Besonders dieser Punkt ist bei der Umsetzung von CloudGrid essenziell, da beim Abarbeiten der Dateioperationen weder der Webserver, noch die Ordnerüberwachung in der Ausführung unterbrochen werden darf. Datenbankzugriffe erfolgen an zwei Stellen im Quellcode umgesetzt. Einerseits beim Initialisieren der Anwendung, wobei die Datenbank angelegt wird, falls diese noch nicht existiert, und zweitens beim Dateihandling. Bei letztgenannten wird beim Starten der Anwendung geprüft, ob es Veränderungen am Dateisystem gab. Sollte dies der Fall sein, werden daraufhin die Hashwerte der Dateien mit denen aus der Datenbank verglichen. Wenn diese nicht übereinstimmen, werden die entsprechenden Dateioperationen erneut durchgeführt. Zusätzlich wird nach dem Upload einer Datei, deren Dateiinformationen in die Datenbank gespeichert.

**underscore:** underscore.js ist eine schlanke aber überaus hilfreiche Open Source Funktionsammlung, welche unter der MIT Lizenz steht, die grundlegende JavaScript Hilfsfunktionen nachliefert. Sie ist an Prototype.js<sup>6</sup> und Ruby<sup>7</sup> angelehnt, ohne dabei bestehende Frameworks vorauszusetzen oder JavaScript Objekte durch Prototyping zu erweitern. Gut die Hälfte aller Funktionen beziehen sich auf Arrays und Objekte. So werden Funktionalitäten wie beispielsweise das Durchlaufen eines Arrays mit »\_.each« oder Filtern von Einträgen eines Arrays oder auch Objektes realisiert. Moderne Browser unterstützen bereits einige dieser Funktionen, sodass in diesem Fall auf selbige zurückgegriffen wird. Durch die Verwendung der underscore.js eigenen Funktionen wird jedoch eine Abwärtskompatibilität für ältere Browser geschaffen.

**watchr:** Node.js ist bereits mit einer eigenen Dateisystemüberwachung ausgestattet. Jedoch weist diese einige Probleme auf. So wird dieses Modul von Node.js selbst als „unstable“

---

<sup>6</sup><http://prototypejs.org>

<sup>7</sup><http://www.ruby-lang.org/de>

bezeichnet, was bedeutet, dass dieses Modul nicht zwingend abwärtskompatibel ist und die durchgeführten Test nicht für eine stabile Version ausreichen. Darüber hinaus ist kein rekursives Überprüfen von Unterordnern möglich.

Abhilfe schafft da das Modul »watchr«. Dieses normalisiert die API von Node.js, sodass es auch zu früheren Version abwärtskompatibel ist. Weiterhin ermöglicht es das rekursive Überwachen von Dateien und Ordnern, was für CloudGrid unabdingbar ist. Aktuell gibt das Modul beim Anlegen, Bearbeiten und Löschen einer Datei oder eines Ordners detaillierte Informationen an den Entwickler zurück. Dieser kann das entsprechende Event entgegennehmen und darauf reagieren. In einer zukünftigen Version wird auch das Event »umbenennen« integriert, welches momentan nur mit einem Workaround realisierbar ist. »watchr« ist ein wichtiger Bestandteil der Anwendungslogik und deckt die Anforderungen für CloudGrid weitestgehend ab.

## 5.3 Selbst entwickelte Klassen und Funktionssammlungen

In diesem Abschnitt wird auf die relevanten selbst entwickelten Klassen und Funktionssammlungen eingegangen. Dabei soll deren grobe Funktionsweise und Anwendungsgebiet aufgezeigt werden.

**filehelper:** Der »filehelper« ist eine Funktionssammlung, welche Methoden für Dateioperationen beinhaltet, unter anderem jene, die vor dem Upload einer Datei auf selbige anzuwenden sind. Dazu zählt das Komprimieren, Verschlüsseln und Splitten einer Datei, sowie deren Umkehrung, also das Dekomprimieren, Entschlüsseln und Zusammenfügen. Weiterhin existieren Funktionen, welche es ermöglichen, alle Dateien eines Ordners zu löschen und ein Array von Dateien zu löschen. Erstgenannte wird benötigt, um den Temp-Ordner beim Start der Anwendung zu leeren. Letztere löscht alle temporär erstellten Dateien nach dem Upload einer Datei. Nicht zuletzt wurde eine Funktion entwickelt, welche die Checksumme einer Datei zurück gibt. Diese wird in CloudGrid in die Datenbank gespeichert und nach dem Download einer Datei verglichen, um sicherzustellen, dass die Datei konsistent ist.

**infologger:** Die Klasse »infologger« dient zum Speichern von Ereignissen während der Ausführung von CloudGrid. Dazu wird im versteckten Ordner ».logging« für jeden Tag eine Datei angelegt, in der Informationen gespeichert werden. Der Benutzer hat in der GUI die Möglichkeit, sich die Einträge der Datei ausgeben zu lassen. Es gibt drei Methoden, »log« , »error« und »warning«, zum Schreiben von Einträgen. Hierbei wendet der Programmierer, je nach Ereignis, die entsprechende Methode an. Alle Einträge beginnen mit einem Datum, gefolgt von einer Tilde und dem entsprechenden Text. Sollte die Methode error verwendet werden, wird dem Text das Wort »error:« vorangestellt und bei warning das Wort »warning:«. Diese Klasse wird global in CloudGrid eingebunden, sodass diese in allen Modulen verfügbar ist und verwendet werden kann.

**initialize:** Beim Starten von CloudGrid müssen diverse Funktionen ausgeführt werden, welche die Konsistenz der Daten sicherstellen. Beispielsweise wird der »temp« Ordner geleert, sodass nicht unnötig Speicherplatz belegt wird. Die JSON Konfigurationsdateien werden auf

ihre Existenz geprüft und im Falle eines Nichtvorhandenseins neu angelegt, genau wie die SQLite Datenbank. Darüber hinaus wurden Funktionen erstellt, welche beim ersten Start der Anwendung direkt nach der Installation zum initialen Setup dienen. Außerdem befinden sich die Methoden zum Überprüfen des Dateisystems auf Veränderungen seit dem letzten Ausführen der Anwendung, sowie der Überprüfung der Dateien auf den Cloudservices in der Klasse. Dementsprechend braucht die Klasse relativ viel Zeit zum Ausführen, was die erste Nutzung der Anwendung verzögert. Dieser Aspekt bedarf einer Optimierung und sollte in zukünftigen Versionen angepasst werden.

**multipart:** Die »multipart« Klasse dient zum Erstellen von HTTP Multipart Bodies.<sup>8</sup> Diese werden unter anderem beim Upload von Dateien benötigt. Quellcode 4 zeigt beispielhaft einen solchen auf.

Ein Multipart Body wird in mehrere Abschnitte unterteilt. Einerseits muss ein entsprechender HTTP Body an den Server mitgegeben werden, damit dieser den Request entsprechend abarbeiten kann, andererseits müssen die entsprechenden Daten übergeben werden. Diese können wiederum im Klartext oder als Binärdaten übergeben werden. Letzteres dient dabei der Übertragung von Dateien an einen Server. Alle Formularfelder durch einen eindeutigen Schlüssel, der Boundary heißt, von einander getrennt. Dies ist ein eindeutiger Schlüssel, der die einzelnen Abschnitte untergliedern soll. Hierbei ist die Eindeutigkeit des Schlüssels ein zwingendes Kriterium. Sollte dieser im Datenbereich eines Formularfeldes auftreten, kann der ganze Multipart Body ungültig werden.

```
1 POST /path/to/script.php HTTP/1.0
2 Host: example.com
3 Content-type: multipart/form-data, boundary=AaB03x
4 Content-Length: body_length
5
6 --AaB03x
7 content-disposition: form-data; name="field1"
8
9 Test Daten
10 --AaB03x
11 content-disposition: form-data; name="userfile"; filename="$filename"
12 Content-Type: application/pdf
13 Content-Transfer-Encoding: binary
14
15 binäre Daten
16 --AaB03x--
```

Quellcode 4: Beispiel eines HTTP Multipart Bodies

Die selbsterstellte Bibliothek »multipart« greift darüber hinaus auf die externe Bibliothek namens »mime« zu, welche es erlaubt, den MIME Type einer Datei programmatisch zu bestimmen, um somit den Content-Type Header zu befüllen. »multipart« wird nach Beendigung dieser Arbeit als Open-Source Module in den Node.js eigenen Paketmanager npm eingepflegt, da zum Zeitpunkt der Erstellung dieser Arbeit keine vergleichbaren Module existierten oder deren Lösungen als unzureichend erachtet wurden.

---

<sup>8</sup><http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4.2>



**oauth2:** Die oauth2 Klasse basiert auf einem Node.js Modul Namens »oauth«<sup>9</sup>. Dieses steht unter der MIT Lizenz, sodass es möglich war, diese nach Belieben anzupassen. Grundsätzlich dient das Modul dazu generelle Aufgaben bei der Verwendung von OAuth2 abzudecken. Unter anderem kann der Authentifizierungstoken von einem Anbieter abgerufen werden und POST und GET Requests an diesen geschickt werden. Das Modul hat bereits viele Anforderungen abgedeckt, jedoch war es für die Umsetzung des Prototypen nicht ausreichend, sodass dieses noch um weitere Funktionalitäten erweitert werden musste. So ist es nun möglich neben POST und GET Request auch DELETE und PUT Request zu erstellen. Dies ist beispielsweise für die Anbieter Dropbox und Box zwingend notwendig, da diese Anbieter beide Methoden in ihrer API integriert haben.

Weiterhin wurde der Prozess zum Erstellen des Request Bodys angepasst, sodass auch ein Node.js Buffer oder gar ein Array von Buffern anstelle eines Strings an den Body übergeben werden kann und in den entsprechenden Body geschrieben wird. Neben diesen Anpassungen wurde zudem der Code refactored, was bedeutet, dass einige syntaktische Fehler behoben wurden und der Quellcode besser strukturiert wurde, was die Lesbarkeit erheblich steigert. Bei diesem Prozess wurden auch mehrere kleinere Fehler ausfindig gemacht, welche zugleich beseitigt wurden. In Anschluss an diese Arbeit wird auch dieses Modul, genau wie das »multipart« Module als Open-Source Module in das npm eingepflegt.

**queue:** »queue« ist eine Hilfsklasse, welche es ermöglicht, lang andauernde Funktionen nacheinander auszuführen. Besonders bei der Verwendung der SQLite Datenbank ist dieses unerlässlich, da diese nur einen Schreibzugriff zur gleichen Zeit zulässt. Das bedeutet beispielsweise, wenn die Teilstücken einer Datei zu den Cloudservices hochgeladen wurden und die Information darüber in die Datenbank geschrieben werden soll, muss sichergestellt werden, dass die Schreibzugriffe nacheinander ausgeführt werden. Sonst würde das Programm mit einer Fehlermeldung abstürzen.

Die Klasse selbst lehnt sich an dem »reference counting« Prinzip an. Dieses besagt, dass eine Queue, also eine Warteschlange, erstellt und bei jedem Eintrag der hinzugefügt wird, ein klasseninterner Zähler inkrementiert und beim Entfernen eines Eintrages aus der Queue, der Zähler dekrementiert wird. Solange wie der Zähler nicht auf 0 hinuntergeht, wird die Klasse weiter ausgeführt. Sollte er auf Null stehen, so wird auf weitere Einträge gewartet.

Um dieses Prinzip umzusetzen, wird ein Array verwendet, welches die einzelnen Funktionen der Queue beinhaltet. Mit der Methode »push« kann ein weiteres Element der Queue hinzugefügt werden. Ein boolescher Parameter »inProgress« gibt an, ob zum Zeitpunkt des Hinzufügens eine Funktion ausgeführt wird oder ob direkt mit der Ausführung der aktuellen Funktion begonnen werden kann. Sollte keine Funktion ausgeführt werden, so wird die Methode »next« aufgerufen. Diese setzt den Parameter inProgress auf true und nimmt sich daraufhin den obersten Eintrag des Arrays und führt die Funktion aus. Sollte kein Eintrag mehr im Array existieren, also nach dem Prinzip des »reference counting« der Zähler auf Null stehen, so wird der inProgress auf false gesetzt.

Eine Funktion, die an die queue Klasse übergeben wird, muss als Callback die next Methode aufrufen, damit die Funktionalität der Klasse gewährleistet werden kann. Diese Limitierung ist speziell auf die Verwendung von Node.js zugeschnitten und ist ein gewünschtes Verhalten. Sollte der Programmierer diese Methodik nicht implementieren, so wird die Queue nicht weiter

---

<sup>9</sup><https://npmjs.org/package/oauth>

abgearbeitet. Im Falle eines Fehlers bei der Ausführung einer Datei, wird dieser abgefangen und die nächste Funktion aufgerufen.

**randomhelper:** Eine weitere unterstützende Funktionssammlung ist der »randomhelper«. Dieser ermöglicht es, verschiedene pseudozufällige Werte zu erzeugen. So wird beispielsweise die Methode »generateFileName« verwendet, um pseudozufällige Dateinamen für die Teilstücke einer Datei zu generieren. Die Methode »generatePassphrase« dient hingegen dazu, einen Passphrase zur Ver- und Entschlüsselung einer Datei zu liefern.

**watchrwrapper:** In der Klasse »watchrwrapper« wird ein Großteil der Anwendungslogik von CloudGrid umgesetzt. Alle Dateioperationen werden unter Zuhilfenahme, sowohl der externen Module als auch selbstentwickelter Klassen und Funktionssammlungen, in dieser Klasse umgesetzt. Zuerst wird das Modul »watchr« gestartet, welches in Abschnitt 5.2 bereits vorgestellt wurde. Dieses überprüft einen, vom Benutzer gewählten Ordner und dessen Unterordner, auf Veränderungen. Je nach Ereignis wird daraufhin der entsprechende Prozess, »createProcess«, »updateProcess« und »deleteProcess« genannt, gestartet. Diese laufen in einer Queue ab, was bedeutet, dass diese synchron nacheinander ausgeführt werden. Nach Beendigung eines Prozesses wird entweder der nächste gestartet, insofern einer in der Queue existiert, oder die Klasse wartet auf die nächste Veränderung.

**provider:** Die Klassen zur Implementierung der einzelnen Cloudservices werden in einem Ordner »provider« gespeichert. Alle Klasse verfügen über einen ähnlichen Funktionsumfang und wurden in Methodenbenennung und Aufbau einheitlich gestaltet. Lediglich Besonderheiten der einzelnen Provider müssen in den Methoden beachtet werden, was eine einheitliche generische Klasse für alle Anbieter nicht realisierbar macht. Jede Klasse bekommt im Konstruktor ein Objekt mit Einstellungen übergeben. Diese werden aus der »provider.json« Datei geladen. Ein späteres Hinzufügen und Auslesen ist durch entsprechende Getter und Setter jedoch ebenfalls möglich. Weiterhin bindet der Konstruktor die »OAuth2« Klasse ein und wird ausgeführt. Je nach Anbieter wird dann noch geprüft ob ein Access-Token existiert und ob dieser noch gültig ist. Sollte dem nicht der Fall sein, wird entsprechend mit dem Refresh-Token ein neuer angefordert. Danach ist die Klasse einsatzbereit. Alle Methoden werden unter Zuhilfenahme der »oauth2« Klasse ausgeführt und entsprechend der Dokumentation des Anbieters umgesetzt. Jedes Modul wurde wiederverwendbar entwickelt. In Anschluss an diese Arbeit wird auch bei den Provider Modulen erwägt, diese als Open-Source Module in das npm einzupflegen, da es zum Zeitpunkt der Erstellung dieser Arbeit keine Module für Box, Microsoft Skydrive oder Google Drive existierten.

## 5.4 Realisierung der Datenhaltungs-Schicht

Die physische Datenhaltung wird durch zwei Konzepte realisiert. Durch die Speicherung einfacher JavaScript Strings in einer JSON Datei werden alle Konfigurationseinstellungen gespeichert. Dazu zählen die Einstellungen zu den Cloudservices, welche exemplarisch im Quellcode 5 aufgeführt sind, sowie die Systemeinstellungen und benutzerspezifischen Einstellungen, dazu zählen der überwachte Ordner, die Anwendungssprache und der Port des Webservers.

```
1 {  
2   "box": {  
3     "id": 1,  
4     "client_id": "506o7ezrhikyx35bwcuyzzg2zxxx",  
5     "client_secret": "KMWtKieXazyRHtQXzxrOC6JlClxxx",  
6     "base_site": "https://www.box.com/",  
7     "authorize_url": null,  
8     "access_token_url": "api/oauth2/token",  
9     "custom_headers": null,  
10    "access_token": "SAuJtuY7VrBd3ZGj3lPOKaJj7qQxxx",  
11    "refresh_token": "h2G9BFxcsvpoklapuDvtf0uky0haTAxxx",  
12    "expires": 1374799440,  
13    "redirect_uri": "http://127.0.0.1:8080/auth/box/authorize",  
14    "connected": true  
15  }  
16 }
```

Quellcode 5: JSON String eines Cloudservices

Jegliche JSON Dateien werden bei dem Anwendungsstart von der selbst entwickelten Bibliothek »jsonreader« eingelesen und in einer globalen Variable vorgehalten. Diese ist auch in allen Submodulen verfügbar, sodass die Anwendung jederzeit Zugriff auf diese Informationen hat. Der Vorteil daran ist, dass die Datei lediglich einmal beim Anwendungsstart eingelesen werden muss und nicht jedes Mal, wenn diese benötigt wird. Sollten sich Daten ändern, so wird die komplette Datei neu geschrieben. Da Änderungen an den Einstellungen selten vorkommen und die zu speichernden Daten einen geringen Umfang haben, ist dieses Verfahren überaus performant.

Bei der Wahl des Formats zum Speichern der Informationen wurde JSON gewählt, da es, wie der Name bereits erahnen lässt, von Hause aus in JavaScript implementiert und dementsprechend leicht umsetzbar ist. Die Daten werden aus der Datei ausgelesen und in ein JavaScript Object umgewandelt. Danach ist es möglich Attribute, ähnlich wie in einer Klasse, direkt anzusprechen. Gerade bei verschachtelten Strukturen ist es somit leichter möglich auf Attribute zuzugreifen.

Die zweite verwendete Methode ist die Speicherung der Daten in einer SQLite Datenbank. Diese hat den Vorteil, dass sie sich direkt in eine Anwendung implementieren lässt und keine weitere Server-Software benötigt wird. Erforderlich ist lediglich die Einbindung einer entsprechenden Bibliothek. Im konkreten Fall von CloudGrid wird das »sqlite3« Modul verwendet. Die Datenbank selbst liegt im Ordner »models« und heißt config.db. Diese enthält lediglich zwei Tabellen. In Abbildung 11 sind diese schematisch aufgezeigt.

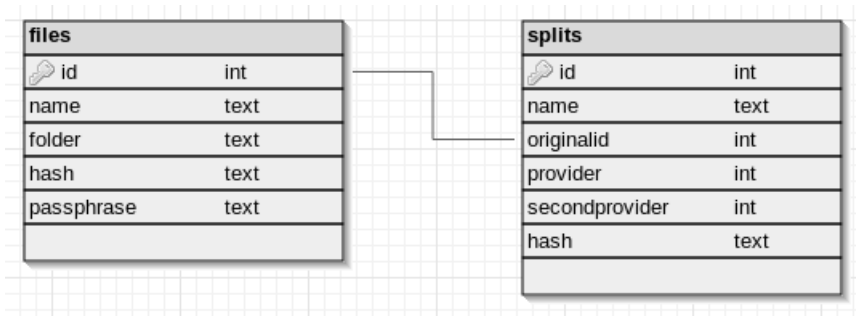


Abbildung 11: Schema der SQLite Datenbank

Die Tabelle »files« speichert Informationen zur Originaldatei, den Namen, den Ordner in dem diese auf dem lokalen Dateisystem gespeichert ist, sowie den Hash des Inhalts und den Passphrase zum Ent- und Verschlüsseln der Datei. Der Hash wird benötigt um nach dem Download einer Datei und der Durchführung der Dateioperation zu prüfen, ob die Datei korrekt wiederhergestellt wurde.

In Tabelle »splits« werden die Informationen zu den Teilstücke einer Datei gespeichert. Neben dem Namen und der Verknüpfung zur ursprünglichen Datei, hier »originalid« genannt, wird ein weiterer Hash gespeichert und die Provider, bei denen die Datei geuploaded wurde. Der Wert »providerid« entspricht der ID aus der JSON Datei, welche im Quellcode 5 aufgeführt ist. Somit können programmatisch die Teilstücke von den einzelnen Anbietern geladen werden. Nachdem der Download beendet ist, werden die Teilstücke mit den in der Datenbank gespeicherten Hashwerten verglichen, um auch in diesem Fall die Dateiintegrität sicherzustellen.

## 5.5 Realisierung der Anwendungslogik-Schicht

Der Workflow der Anwendungslogik-Schicht beginnt mit dem Initialisieren der Anwendung. Hierbei wird als Erstes die Konsistenz der benötigten Ordner überprüft. Dazu zählen der temp, logging und models Ordner. Sollte einer dieser drei Ordner nicht existieren, wird er von der Anwendung angelegt. Daraufhin werden die JSON Dateien im models Ordner überprüft. Sollten auch diese nicht vorhanden sein, werden sie angelegt und mit Standardwerten gefüllt. Als weiterer Schritt wird geprüft, ob die SQLite Datenbank existiert. Sofern sich auch diese nicht im models Ordner befindet, wird sie angelegt und die beiden Tabellen erzeugt. Abschließend wird der temp Ordner geleert, für den Fall, dass sich noch temporäre Dateien in diesem befinden. Dieser Vorgang wird sowohl beim ersten Starten der Anwendung durchgeführt, als auch bei jedem weiteren Start. Eine Unterscheidung dieser beiden Vorgänge wird nicht getroffen.

Wenn sich die Daten der Anwendung daraufhin in einem konsistenten Zustand befinden, werden alle Einstellungen geladen. Die Daten aus den JSON Dateien werden gelesen und in einer globale Variable im System hinterlegt. Das ermöglicht bei der Ausführung von CloudGrid einen schnelleren Zugriff auf diese Daten und vermindert die Anzahl der Zugriffe auf das Dateisystem.

Sobald auch dieser Vorgang abgeschlossen ist, wird parallel der Webserver und die Ordnerüberwachung gestartet. Auf Seiten des Webserver werden dazu alle Einstellungen geladen,

wie beispielsweise der Port, auf dem die Webseite laufen soll, die Templateengine wird festgelegt und die einzelnen Standardtemplates vorgeladen. Zusätzlich werden alle Routen definiert. Außerdem werden die gespeicherten OAuth-Token der Cloudservices auf Aktualität geprüft. Sollte der access token eines Services abgelaufen sein, so wird dieser, mit Hilfe des refresh tokens, erneuert.

Die Initialisierung der Ordnerüberwachung erfolgt in mehreren Schritten. Als Erstes wird das aktuelle Dateisystem auf Veränderungen überprüft. Dazu werden, mit der selbsterstellten Klasse »foldertree«, alle Dateien erfasst und deren Hashwert ermittelt, um diese daraufhin mit dem Hashwert in der Datenbank zu vergleichen. Sollten die diese nicht übereinstimmen, so werden zuerst die Teilstücke der Datei bei den Cloudservices gelöscht, um danach die Dateioperationen auf die Datei anzuwenden und diese anschließend auf die Cloudservices zu verteilen. Wenn jedoch eine Datei nicht in der Datenbank existiert, so wird diese in die Datenbank aufgenommen, die Dateioperationen angewendet und entsprechend zu den Cloudservices hochgeladen. Abschließend werden gelöschte Dateien ermittelt, das bedeutet, alle Dateien welche sich in der Datenbank befinden, jedoch nicht auf dem Dateisystem. Wenn dieser Fall eintritt, werden sowohl die Teilstücke auf den Cloudservices, als auch die Informationen über die Datei aus der Datenbank gelöscht.

Wenn dieser Vorgang beendet ist, wird die Konsistenz der Teilstücke bei den Cloudservices überprüft. Dazu werden die Informationen zu den Teilstücken aus der Datenbank ausgelesen und selbige auf Existenz bei den Cloudservices geprüft. Wenn ein Teilstück fehlen sollte, wird versucht dieses bei dem redundanten Cloudservices herunterzuladen und erneut hochzuladen. Sollte das Teilstück bei allen Cloudservices gelöscht worden sein, so werden alle Teilstücke der originalen Datei auf den Cloudservices gelöscht, sowie die Informationen in der Datenbank, um die Datei daraufhin wie eine Neuanlage zu betrachten und den bereits beschriebenen Vorgang durchzuführen.

Nach der erfolgreichen Durchführung dieses Vorganges wird letztendlich die Ordnerüberwachung gestartet. Diese überwacht einen vom Benutzer angegebenen Ordner auf Veränderung. Dabei können die in Abschnitt 4.3.2 beschriebenen Dateioperationen, anlegen, löschen, bearbeiten, umbenennen und verschieben einer Datei, auftreten. Jeder dieser Fälle muss einzeln abgearbeitet werden. Um sicherzustellen, dass die einzelnen Funktionen erfolgreich durchgeführt werden können und sich beim Zugriff auf die Datenbank nicht gegenseitig blockieren, werden sie, unter Zuhilfenahme der queue Klasse, in eine Warteschlange eingefügt. Diese führt alle dort enthaltenden Funktionen nacheinander aus, sodass keine gleichzeitigen Zugriffe auftreten können.

**Anlegen:** Beim anlegen einer Datei wird die Methode »createProcess« ausgeführt. Diese prüft zuerst, ob die angegebene Datei wirklich existiert und ob es sich um eine Datei oder einen Ordner handelt. Sollte die Datei nicht existieren oder ein Ordner übergeben worden sein, so bricht die Methode mit einer Fehlermeldung ab. Ansonsten beginnt sie mit der Komprimierung der Datei. Dabei wird eine Funktion aus dem filehelper verwendet. Zum Komprimieren wird die Node.js eigene »zlib« Klasse verwendet. Eine Datei, die komprimiert wurde, wird in dem temp Ordner der Anwendung gespeichert und mit der Dateiendung »zip« versehen. Sollte dieser Vorgang erfolgreich gewesen sein, so wird ein Passphrase erstellt und die Datei daraufhin verschlüsselt. Auch in diesem Fall wird die Datei im temp Ordner gespeichert und mit der Dateiendung »crypt« versehen. Nach der erfolgreichen Verschlüsselung wird die Datei gesplittet. Dazu wird zuerst die Anzahl der eingebundenen Cloudservices ermittelt, um an-

hand dieser die Anzahl der Teilstücken zu bestimmen. Beim Splitten wird jedes Teilstück mit einem pseudozufälligen Dateinamen versehen und im temp Ordner gespeichert. Im Anschluss daran werden die eingebundenen Cloudservices ausgewählt und der Upload der Dateien wird durchgeführt. Abschließend werden alle Informationen zur originalen Datei und zu den Teilstücken in der SQLite Datenbank gespeichert und alle temporären Dateien aus dem temp Ordner gelöscht. Sollte in dem gesamten Prozess ein Fehler auftreten, so wird dieser unter Zuhilfenahme der »infologger« Funktionssammlung in eine Logging Datei geschrieben und kann vom Benutzer in der GUI eingesehen werden. Der Prozess selbst wird abgebrochen und der nächste Prozess gestartet, insofern es einen weiteren in der Queue gibt.

**Löschen:** Beim Löschen einer Datei wird die Methode »deleteProcess« ausgeführt. Diese ermittelt zuerst die Datenbank ID der originalen Datei, um daraufhin alle Teilstücke zu selektieren. Sobald diese Informationen vorliegen, werden alle Teilstücke bei allen Cloudservices gelöscht. Dazu werden die entsprechenden Methoden in den Provider Klassen ausgeführt. Abschließend werden die Einträge in den Datenbanken gelöscht. Auch die deleteProcess Methode wird durch die infologger Funktionssammlung protokolliert und an die Queue übergeben.

**Bearbeiten:** Die Bearbeitung einer Datei ähnelt einer Vereinigung der beiden zuvor vorgestellten Prozesse. Sollte sich der Inhalt einer Datei verändern, werden zuerst alle Teilstücke einer Datei auf den Cloudservices und zugleich deren Informationen aus der Datenbank gelöscht. Die Informationen zu der originalen Datei bleiben in der Tabelle files in der Datenbank erhalten. Daraufhin wird der Hash der Datei neu berechnet und der Datenbankeintrag entsprechend geupdatet. Anschließend werden die Dateioperationen, welche auch im »createProcess« durchgeführt wurden, ausgeführt. Die Teilstücke werden dann zu den Cloudservices hochgeladen und die Informationen in die Datenbank geschrieben.

**Umbenennen/Verschieben:** Zum Zeitpunkt der Erstellung der Arbeit bietet das Modul »watchr« keine Methoden zum Erkennen einer Umbenennung oder Verschiebung einer Datei. Dadurch ergibt sich ein großer Nachteil bei der Implementierung in CloudGrid. Normalerweise würde lediglich der Dateiname beziehungsweise der Pfad zur Datei in der Datenbank angepasst werden und die Teilstücke einer Datei bei den Cloudservices könnten unverändert bleiben. Durch das Fehlen der Methode wird das Umbenennen und auch das Verschieben einer Datei als Löschen und anschließend Erstellen einer Datei angesehen. Das beeinträchtigt erheblich die Effizienz von CloudGrid. Dieser Umstand muss in einer späteren Version des Prototypen angepasst werden, um unnötige Datei- und Uploadoperationen zu vermeiden und die Effizienz bei der Ausführung der Anwendung zu steigern.

Die Ordnerüberwachung läuft solange eine Internetverbindung besteht. Vor der Überwachung wird eine Funktion gestartet, welche im Sekundentakt prüft, ob eine Verbindung zum Internet besteht oder nicht. Sollte diese unterbrochen werden, so wird auch die Ordnerüberwachung gestoppt und die Queue mit den Prozessen geleert. Dieser Vorgang wird durch die Funktion »hasInternet« der »utils« Funktionssammlung realisiert. Um die Überprüfung umzusetzen, wird ein Datenpaket mittels Ping an google.de geschickt. Sollte ein Datenpaket zurückgeschickt werden, besteht eine Internetverbindung, ansonsten nicht. Wenn die Verbindung zum Internet wieder aufgebaut wurde, wird automatisch die Ordnerüberwachung gestartet und der zuvor aufgezeigte Vorgang erneut durchgeführt. Dadurch, dass die initiale Überprüfung der

Dateien durchgeführt wird, werden auch Prozesse, die zuvor aus der Queue gelöscht wurden, erneut ausgeführt.

## 5.6 Realisierung der Präsentationsschicht

Die Umsetzung der Präsentationsschicht ist bewusst einfach gehalten. Zur Gestaltung wurde CSS verwendet, welches aus Syntactically Awesome Stylesheets (Sass) Dateien generiert wird. Sass ist eine Scriptsprache, welche eine programmatische Erstellung von CSS Dateien ermöglicht. Insbesondere die Verwendung von Variablen und Funktionen, Mixins genannt, erleichtern die Arbeit mit CSS erheblich, wodurch das Stylesheet sich später flexibler anpassen lässt. Eine Sass Datei muss nach der Erstellung kompiliert werden, was der mitgelieferte Sass-Compiler realisiert.

Bei der Umsetzung des Designs für die GUI werden vier CSS Dateien verwendet. Die »normalize.scss« dient dazu, die Voreinstellungen von HTML Elementen in verschiedenen Browsern zurückzusetzen. Hierbei wird das normalize Projekt<sup>10</sup> verwendet. Diese steht unter der MIT Lizenz und kann somit frei verwendet werden.

Die zweite Datei »mixins.scss« ist eine selbsterstellte Funktionssammlung, welche sich wiederholende Style-Definitionen umsetzt. Beispielsweise dient das Mixin »clearfix« dazu, gefloatete Elemente zu clearen oder das »center-page« Mixin, um den Header, Footer und Contentbereich in der Seite zu zentrieren.

Die »variables.scss« Datei beinhaltet alle Variablen, welche in den Sass Dateien verwendet werden. Der Vorteil hierbei ist, dass das Farbkonzept der Seite schnell bearbeitet werden kann, da alle Farben in entsprechende Variablen ausgelagert wurden. So können Themes angelegt werden, ohne das eine komplette CSS oder Sass Datei refactored werden muss.

Abschließend existiert noch die »main.scss« . Diese beinhaltet alle Styleangaben für die einzelnen Elemente der Webseite. Jede der zuvor genannten Dateien wird am Anfang der »main.scss« eingebunden, sodass beispielsweise auf alle Variablen oder Mixins zugegriffen werden kann.

Die Umsetzung des Grundgerüsts der Webseite erfolgt nach dem im Abschnitt 4.3.3 erstellten Wireframe. Alle dort angedachten Konzepte wurden umgesetzt. Wohingegen das Farbkonzept minimalistisch gehalten wurde, was bedeutet, dass lediglich Weiß, ein heller Grauton und ein dunkler Grauton verwendet werden. Als akzentuierende Farbe für Buttons und Links wird hingegen ein Blauton verwendet. Das Logo von CloudGrid, welches in Abbildung 12 aufgezeigt wird, wurde selbst entwickelt und soll ein stark abstrahiertes Grid-Cluster aufzeigen, wobei die einzelnen Nodes entsprechend farblich getrennt werden. Somit soll die Dateiverteilung, welche in CloudGrid erfolgt, aufgezeigt und zugleich die Speicherung bei den unterschiedlichen Cloudservices erkenntlich gemacht werden.

---

<sup>10</sup><https://github.com/necolas/normalize.css>



Abbildung 12: Das Logo von CloudGrid

Templates wurden, wie bereits in Abschnitt 4.3.3 beschreiben, mit der Hogan.js Templateengine umgesetzt. Dazu werden generelle Bausteine in dem Unterordner »templates« gespeichert. Hier existieren zwei Bausteine, einer für den Header der Seite, demnach der komplette Bereich bis zum Content und der Footer, welches nach dem Contentbereich beginnt. Im Header werden alle benötigten CSS Dateien eingebunden, sowie das Menü der Seite. Wohingegen im Footer alle JavaScript Dateien eingebunden werden. Das hat den Vorteil, dass die Ladezeit der Seite minimiert wird, da die verminderte Anzahl von HTTP Request vor der Darstellung der Seite vermindert werden.

Neben dem Design wurde auch der Aufbau der einzelnen Seiten schlicht gehalten, damit der Benutzer sich leicht in die GUI von CloudGrid einarbeiten kann. Die Seite »Informationen« verfügt neben der obligatorischen Überschrift und dem Hilfe-Icon nur über eine Textbox. Diese gibt den Inhalt der Loggingdatei des aktuellen Tages aus. Weiterhin erhält der Benutzer die Möglichkeit das Datum der anzuzeigenden Datei durch einen entsprechenden Button zu verändern.

In der Seite »Einstellungen« kann der Benutzer jegliche personenbezogenen Einstellungen verändern. Dazu zählen beispielsweise der Port, auf dem der Webserver läuft, und die ausgewählte Sprache. Beim ersten Start der Anwendung wird eine ähnliche Seite angezeigt, auf der jedoch mehr Einstellungen anpassbar sind. Beispielsweise kann der Ordner bestimmt werden, welcher von CloudGrid überwacht werden soll. Zudem bieten beide Seiten die Möglichkeit, die Authentifizierung für die Cloudservices durchzuführen. Dazu muss lediglich der entsprechende Button des Anbieters angeklickt werden und der in Abschnitt 2.4.4 aufgezeigte Authentifizierungsprozess wird gestartet. Sobald der Benutzer vom Anbieter zurückgeleitet wird, bekommt er eine Informationen über die erfolgreiche Implementierung des Anbieters angezeigt. Zusätzlich wird diese Information in die Logging-Datei geschrieben.



## Kapitel 6

# Evaluation und Demonstration

In diesem Kapitel sollen die Ergebnisse des im Rahmen der Arbeit entwickelten Prototypen, anhand der zuvor definierten Anforderungen und Lösungen, bewertet und vorgestellt werden. Im Anschluss daran wird eine Demonstration des Systems durchgeführt.

### 6.1 Evaluation des Systems

Der Prototyp von CloudGrid wurde ausschließlich auf einem Linux Mint System getestet und ist dort voll lauffähig. Linux Mint basiert auf Ubuntu, sodass die zwei meistgenutzten Linux Distributionen<sup>1</sup> unterstützt werden. Sowohl Windows als auch Mac OS wurden bei der Entwicklung des Prototypen nicht getestet.

Um die GUI korrekt darstellen zu können, werden Browser benötigt, welche HTML5 und CSS3 unterstützen. Dazu zählen Google Chrome, Firefox, Safari und auch Opera, sowie die Internet Explorer ab Version 10. Es kann bei unterschiedlichen Browsern kleinere Abweichungen bei der Positionierung von Elementen geben, welche sich jedoch nicht auf die Funktionalität der GUI auswirken. Dieses Verhalten resultiert aus der unterschiedlichen Interpretation von HTML und CSS der einzelnen Browser.

Ein weiterer Schwerpunkt liegt in der Integration weiterer Cloudservices in CloudGrid. Wenn ein Dienst zur Anwendung hinzugefügt werden soll, müssen dazu drei Anpassungen vorgenommen werden. In den Ordner »provider« muss zuerst eine Klasse integriert werden, welche, entsprechend der Klassen der bestehenden Anbieter, die Funktionalitäten der Clouddienste abbildet. Daraufhin muss der einzubindende Anbieter in der Konfigurationsdatei »provider.json« im Ordner »models« aufgenommen werden. In dieser Datei werden alle Parameter, die bei der Authentifizierung mittels OAuth benötigt werden, hinterlegt. Um abschließend den Anbieter auch in der GUI verfügbar zu machen, muss ein Button in der View »connect.html« hinterlegt werden, so wie die entsprechende Controllerlogik in der »connect.js«. Die Auswahl der Cloudservices beim Upload wird daraufhin automatisch von CloudGrid durchgeführt. Der gesamte Vorgang sollte, in einer späteren Version von CloudGrid, noch modularer und generischer gestaltet werden, sodass ein Entwickler leichter weitere Anbieter integrieren kann.

---

<sup>1</sup><http://distrowatch.com/dwres.php?resource=popularity>

Weiterhin erweist sich die im Systementwurf getätigte Annahme, dass der Dateiupload, weitaus langsamer ist, als die Dateioperationen als richtig. Bei einem Test mit einer 50 Megabyte (MB) großen Datei und lediglich einem Cloudservice, dauerten die Dateioperationen im Durchschnitt 7 Sekunden, wohingegen der Upload der Teilstücke im Durchschnitt rund 2 Minuten dauerte. Diese Werte wurden auf einem Laptop mit einem Intel Core i3 mit 2,4 Gigahertz (GHz), 3 Gigabyte (GB) Arbeitsspeicher, sowie einer Festplatte mit einem Serial Advanced Technology Attachment (SATA) Anschluss und 7200 Umdrehungen pro Minute (upm) ermittelt. Die Uploadgeschwindigkeit wird laut Internetprovider mit 10 mbit/s angegeben. Dieser Engpass kann leider nicht umgangen werden. Vor dem Upload der Teilstücke einer Datei, werden diese bereits komprimiert, um den Vorgang zu beschleunigen. Jedoch reicht das nicht aus, um das Verhältnis auszugleichen. Dateioperationen werden zum jetzigen Stand der Technik schneller durchgeführt, als das Verschicken von Daten über ein Netzwerk oder das Internet. Lediglich durch die Implementierung der, bereits im Abschnitt 5.5 erwähnten fehlenden Unterstützung der Events »Verschieben« und »Umbenennen« bei der Ordnerüberwachung, können unnötige Uploadvorgänge vermieden werden, was Ressourcen einsparen würde. Jedoch bleibt auch dadurch die Grundproblematik bestehen.

## 6.2 Demonstration des Systems

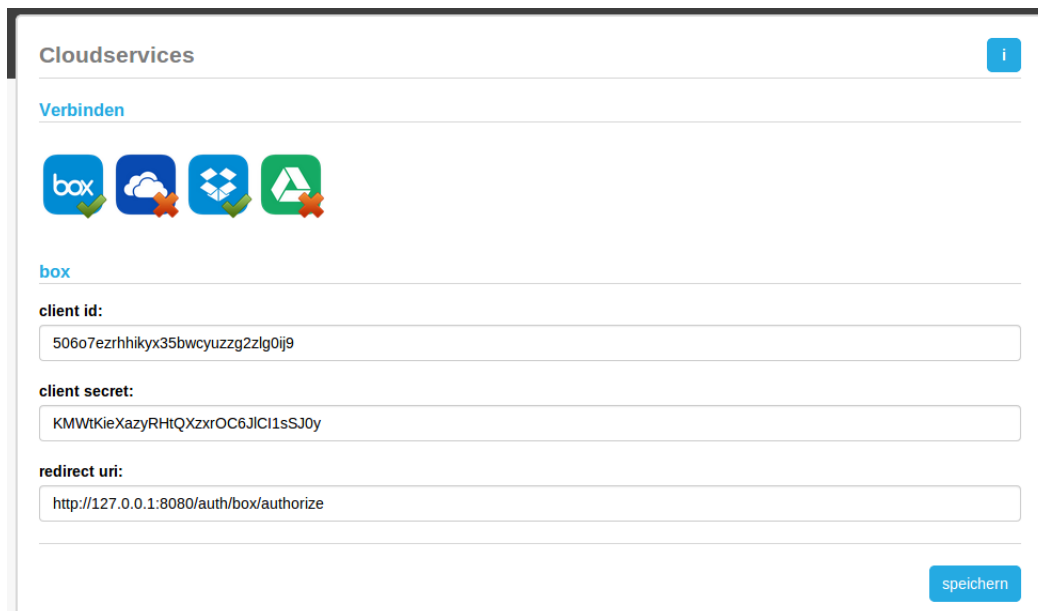
Um die Anwendung zu starten muss die Linux Konsole geöffnet werden und in den entsprechenden Ordner von CloudGrid navigiert werden. Dort muss entweder der Befehl »node CloudGrid.js« oder alternativ die Makefile mittels »make start« ausgeführt werden. Daraufhin ist die GUI im Browser unter <http://localhost:8080> oder alternativ unter <http://cloudgrid.local:8080> erreichbar. Beim ersten Start der Anwendung bekommt der Benutzer eine Einstellungsseite angezeigt, welche in Abbildung 13 zu sehen ist.

The screenshot shows a web interface for 'CloudGrid Einrichten'. At the top, there is a dark navigation bar with four icons and labels: 'Startseite' (house icon), 'Cloudservices' (cloud icon), 'Einstellungen' (gear icon), and 'Informationen' (speech bubble icon). Below this is a white settings form. The form has a title 'CloudGrid Einrichten' with an information icon on the right. It contains three sections: 'Ordner zum überwachen:' with a text input field containing '/home/ueberwacher-ordner'; 'Port des Webservers:' with a text input field containing '8080'; and 'Sprache:' with a dropdown menu showing 'Deutsch'. A blue 'speichern' button is at the bottom right of the form. At the very bottom of the page, there is a dark footer bar with three links: 'Impressum', 'Datenschutzbestimmungen', and 'AGB'.

Abbildung 13: Einstellungsseite beim ersten Start der Anwendung

Hier kann der Benutzer den zu überwachenden Ordner, den Port auf dem der Webserver laufen soll und die Sprache einstellen. Dabei sind alle Werte vordefiniert, sodass der Nutzer sich leichter in diese Seite einfinden kann. Durch Anklicken des blauen »i« Buttons in der rechten oberen Ecke erhält er darüber hinaus Informationen zur Seite. Wenn er die Einstellungen gespeichert hat, wird eine Erfolgsseite angezeigt. Sollte er Felder nicht ausfüllen, erscheint eine entsprechende Fehlermeldung und die Felder können nochmals bearbeitet werden.

Sobald die Speicherung erfolgreich war, muss sich der Benutzer im Menüpunkt »Cloudservices« mit den einzelnen Diensten verbinden. Abbildung 14 zeigt diese Seite auf. Der Benutzer muss zuvor seine »client id«, seinen »client secret« und die »redirect uri« in den entsprechenden Formularfelder hinterlegen, um daraufhin durch einen Klick auf das entsprechende Logo des Anbieters, die Authentifizierung durchzuführen. Er wird daraufhin zum Anbieter weitergeleitet, wo er sich Einloggen muss, um dann CloudGrid die Berechtigung zu geben, auf seine Benutzerdaten zuzugreifen. Abschließend wird er wieder zur Cloudservicesseite zurückgeleitet und erhält einen Hinweis, über die erfolgreiche Verknüpfung mit dem Service. Dienste welche bereits erfolgreich verbunden sind, erhalten einen grünen Haken am Logo, nicht verbundene Dienste ein rotes Kreuz.



The screenshot shows a web interface for managing cloud services. At the top, the title 'Cloudservices' is displayed next to an information icon. Below this, a section titled 'Verbinden' (Connect) contains four service logos. The 'box' logo is marked with a green checkmark, indicating it is connected, while the other three logos (cloud, and two others) are marked with red crosses, indicating they are not connected. Below the logos, there are three input fields for configuration: 'client id' (506o7ezrhikyx35bwcuyzzg2zlg0ij9), 'client secret' (KMWtKieXazyRHtQXzrOC6JlCI1sSJ0y), and 'redirect uri' (http://127.0.0.1:8080/auth/box/authorize). A 'speichern' (save) button is located at the bottom right of the form.

Abbildung 14: Seite zum Verbinden und Bearbeiten der Cloudservices

Wenn auch dieser Vorgang abgeschlossen ist, kann der Benutzer Ordnerüberwachung zum ersten mal starten. Im Header der GUI befindet sich ein Button, welcher betätigt werden muss. Initial ist die Ordnerüberwachung deaktiviert. Abbildung 15 zeigt den Informationstext und den Button auf.

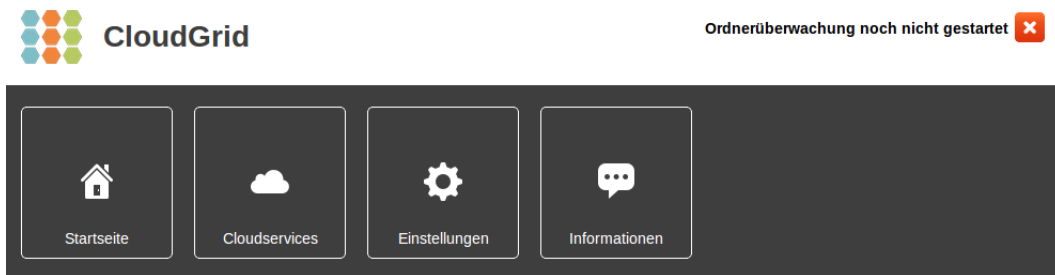


Abbildung 15: Die Ordnerüberwachung ist deaktiviert

Sobald der Benutzer den Button anklickt, wird eine Warteanimation gestartet. Wenn der Vorgang abgeschlossen ist, bekommt er die Information, welche in Abbildung 16 dargestellt ist, angezeigt.

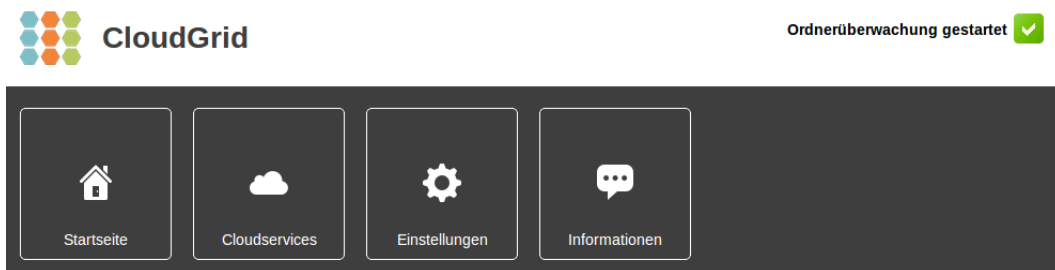


Abbildung 16: Die Ordnerüberwachung wurde gestartet

Die Ordnerüberwachung ist somit gestartet und jegliche Veränderungen werden erkannt. Alle Dateien, welche sich in dem zu überwachenden Ordner befinden, werden zudem beim Start initial eingelesen und zu den Clouddiensten geuploadet. Der Benutzer hat jederzeit die Möglichkeit, die Ordnerüberwachung an- und abzustellen. Dieses Verhalten kann gewünscht sein, wenn beispielsweise größere Veränderungen in dem Ordner durchgeführt werden oder die gesamte Bandbreite der Internetverbindung benötigt wird.

Jegliche Vorgänge kann der Benutzer dabei im Menüpunkt »Informationen« einsehen und den Fortschritt verfolgen. Abbildung 17 zeigt beispielhaft diese Seite auf. Die Einträge in dem Textfeld werden im zwei Sekundentakt aktualisiert, sodass der Benutzer die Seite nicht manuell aktualisieren muss. Zudem kann er sich ältere Einträge mittels der Datumsauswahl oberhalb des Informationsfensters anzeigen lassen.

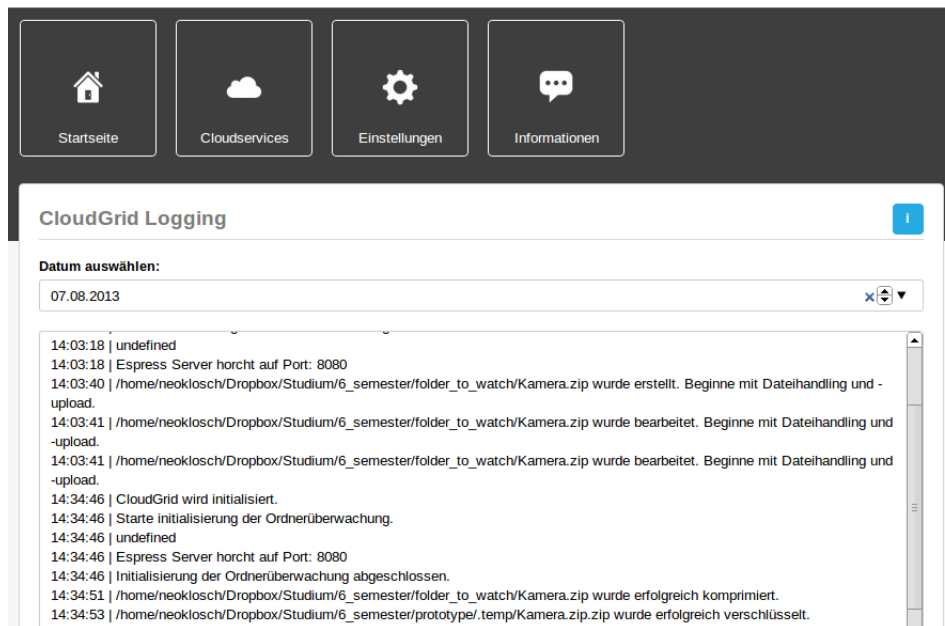


Abbildung 17: Informationsseite von CloudGrid

Die Seite »Einstellungen« ähnelt der Seite, die der Benutzer beim ersten Start der Anwendung angezeigt bekommt. Jedoch sind hier weniger Einstellmöglichkeiten verfügbar. Im Prototypen ist es nicht möglich, den zu überprüfenden Ordner zu wechseln. Das Problem liegt in einer erhöhten Komplexität des Dateihandlings. Es muss geklärt werden, was mit den Dateien im bestehenden Ordner passiert. Dabei können zwei Ansätze verfolgt werden. Im ersten Fall werden alle Dateien in den neu zu überwachenden Ordner kopiert und verbleiben bei den Clouddiensten. Der zweite Fall belässt alle Dateien im ursprünglichen Ordner und löscht die entsprechenden Teilstücke bei den Clouddiensten. Beide Verfahren haben ihre Vor- und Nachteile. Im Prototypen wurde daher lediglich das einmalige Auswählen des Ordner integriert. Jedoch können wie bereits zuvor, sowohl die Sprache, als auch der Port des Webserver bearbeitet werden.

Im Footer der Seite wurden hingegen die drei Seiten »Impressum«, »AGB« und »Datenschutz« aufgenommen. Da diese funktional nicht relevant sind, sind sie momentan mit Blindtext gefüllt, bedingt durch die im Abschnitt 4.2.3 erwähnte Problematik, dass diese inhaltlich durch einen Anwalt erstellt werden müssen.

Letztendlich wurde das in Abschnitt 4.3.3 angedachte Designkonzept komplett umgesetzt und erfüllt funktional alle Anforderungen. Auf allen Seiten ist ein einheitliches Layout zu erkennen, welches mit der »hogan« Templateengine modular umgesetzt wurde.

# Kapitel 7

## Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, einen Prototypen zu entwickeln, der die Möglichkeiten des Dateimanagements unter Verwendung verschiedener Cloudservices aufzeigt.

Um die gestellte Aufgabe zu lösen, wurden in der Anforderungsanalyse die Voraussetzungen für das System dargelegt. Dabei wurde das grobe Konzept des Prototypen erklärt und die unterschiedlichen Anforderungen an die Cloudservices und die zu verwendenden Technologien erarbeitet. Weiterhin dienten die erstellten Use-Cases dem besseren Verständnis der Funktionsweise des Prototypen. Die Evaluation bestehender Anwendung und Dienste zeigt Umsetzungsmöglichkeiten und zugleich Abgrenzungskriterien für CloudGrid auf.

Anschließend wurde im Kapitel Systementwurf konkreter auf die geplante Umsetzung des Prototypen eingegangen. Dabei wurde sowohl die zu verwendende Programmierumgebung ermittelt, als auch die Architektur des Systems aufgezeigt. Diese konnte in drei Schichten, Datenhaltungs-, Anwendungslogik- und Präsentationsschicht, unterteilt werden. Die Evaluation der Cloudservices wertete, sowohl aus technischer, als auch aus rechtlicher Sicht, bestehende Dienste, anhand der zuvor definierten Kriterien, aus.

Die konkrete Umsetzung des Prototypen wurde daraufhin im Kapitel Implementierung ausgearbeitet. Dazu wurde neben der Projektstruktur, die Funktionsweise externer und auch selbst entwickelter Module erklärt und deren Einbindung in das System erläutert. Weiterhin wurden die Realisierung der einzelnen Schichten erörtert.

Abschließend wurde im Abschnitt Evaluation und Demonstration der entwickelte Prototyp bewertet und die Handhabung, sowie die Funktionsweise, aufgezeigt.

Im Prototypen von CloudGrid wurden alle zuvor erarbeiteten Anforderungen umgesetzt. Bei einer zukünftige Weiterentwicklung des vorgestellten Systems sollte primär auf eine Erweiterung des Cloudservice Portfolios gesetzt werden. Um dies zu realisieren, sollten sowohl mehr Authentifizierungsmethoden unterstützt werden, als auch das Datenformat XML, da dies von mehreren Anbietern verwendet wird. Dadurch könnten bereits vier weitere, der in Abschnitt 4.2 evaluierten Anbieter, eingebunden werden. Darüber hinaus würde die Anzahl nochmals steigen, wenn auch kostenpflichtige Anbieter hinzugefügt werden. Jedoch bleibt der Nachteil bestehen, dass anbieterspezifische Funktion nicht in CloudGrid integrierbar sind, wie beispielsweise die Versionierung von Dropbox.

Weiterhin sollte eine Synchronisierung über mehrere Clients realisiert werden. Das erhöht

die Nutzbarkeit für den Anwender und entspricht der Funktionalität aktueller Clientanwendungen der Anbieter. Hierbei müssen wahrscheinlich größere systemarchitektonische Anpassungen vorgenommen werden. Ein Konzept wäre, die lokalen Benutzerdaten von CloudGrid ebenfalls redundant bei den Cloudservices vorzuhalten, um diese auf einem weiteren Client einzubinden. Ebenfalls denkbar wäre die Einbindung eines weiteren Serverdienstes, der die entsprechenden Informationen vorhält. Allerdings ist diese Möglichkeit abweichend von der momentanen Grundidee von CloudGrid, dem Benutzer ein System zu ermöglichen, bei dem er jederzeit Überblick über den Verbleib seiner Daten hat.

Zudem würde die Umsetzung mobiler Anwendungen, für Smartphones und Tablets, die Anwenderfreundlichkeit erhöhen. Auch hier müssten konzeptionelle Änderungen durchgeführt werden, welche sich möglicherweise mit denen des Multiclient Konzepts gleichen.

Auf Seiten der Node.js Anwendung würde die Erweiterung des »watchr« Moduls um die Events »Umbenennen« und »Verschieben« die Performance steigern und unnötige Dateioperationen vermeiden. Weiterhin ist es möglich, weitere Einstellungsmöglichkeiten, wie das Setzen eines anderen oder eines weiteren zu überprüfenden Ordners oder auch die Wahl des Verschlüsselungsalgorithmus. Dabei muss jedoch beachtet werden, dass sich solch eine Option auf bereits hochgeladene Dateien auswirken würde.

Wie bereits in Abschnitt 6.1 beschrieben, sollte auch die Einbindung weiterer Cloudservices modularer und generischer gestaltet werden. Momentan müssen mehrere Dateien bearbeitet werden, um einen Dienst zu CloudGrid hinzuzufügen. Besser wäre es, wenn es einen Ordner geben würde, wo Module für Cloudservices vorgehalten und weitere hinzugefügt werden können. Dieser wird beim Systemstart auf neue Module geprüft und diese entsprechend eingebunden.

Abschließend sollte die GUI erweitert werden, um die Benutzerfreundlichkeit zu steigern. Dazu zählen mehr Informationen für den Benutzer, wie beispielsweise ein Hinweis, wenn der Speicherplatz eines Anbieters ausgereizt ist oder ein Anbieter nicht mehr verfügbar ist.

Letztendlich zeigt das hier vorgestellte System, erfolgreich und in einer prototypischen Qualität, die Funktionsweise des erarbeiteten Konzeptes auf. Das Ergebnis kann somit als solide Grundlage für eine Weiterentwicklung angesehen werden.

# Literaturverzeichnis

- [Ben04] BENDEL, Günther: *Grundkurs Verteilte Systeme*. 3., verbesserte und überarbeitete Auflage. Vieweg + Sohn Verlag, GWV Fachverlage GmbH, 2004. – ISBN 3-528-25738-5
- [BKNT10] BAUN, Christian ; KUNZE, Marcel ; NIMIS, Jens ; TAI, Stefan: *Cloud Computing*. Springer-Verlag, 2010. – ISBN 978-3-642-01593-9
- [Box13a] BOX.COM LTD: *BOX-DATENSCHUTZERKLÄRUNG UND COOKIE-RICHTLINIE FÜR MOBILE UND ONLINE-NUTZUNG*. Online-Quelle, 2013. – [https://app.box.com/legal\\_text/index.php?type=de\\_GB&rm=get\\_legal\\_text&page=privacy\\_policy](https://app.box.com/legal_text/index.php?type=de_GB&rm=get_legal_text&page=privacy_policy); letzter Zugriff: 11.08.2013, 16:17 Uhr
- [Box13b] BOX.COM LTD: *NUTZUNGSBEDINGUNGEN VON BOX*. Online-Quelle, 2013. – [https://app.box.com/legal\\_text/index.php?type=de\\_GB&rm=get\\_legal\\_text&page=tos](https://app.box.com/legal_text/index.php?type=de_GB&rm=get_legal_text&page=tos); letzter Zugriff: 11.08.2013, 16:17 Uhr
- [Dro12] DROPBOX INC.: *Allgemeine Geschäftsbedingungen für Dropbox*. Online-Quelle, 2012. – <https://www.dropbox.com/privacy#terms>; letzter Zugriff: 20.07.2013, 19:43 Uhr
- [Dro13a] DROPBOX INC.: *Dropbox-Datenschutzrichtlinien*. Online-Quelle, 2013. – <https://www.dropbox.com/privacy>; letzter Zugriff: 20.07.2013, 23:41 Uhr
- [Dro13b] DROPBOX INC.: *Überblick über Sicherheitsfunktionen*. Online-Quelle, 2013. – <https://www.dropbox.com/privacy#security>; letzter Zugriff: 21.07.2013, 00:03 Uhr
- [Fuj10] FUJITSU RESEARCH INSTITUTE AND PRODUCED BY FUJITSU GLOBAL BUSINESS GROUP: *Personal data in the cloud: A global survey of consumer attitudes*. Online-Quelle, 2010. – [http://www.fujitsu.com/downloads/SOL/fai/reports/fujitsu\\_personal-data-in-the-cloud.pdf](http://www.fujitsu.com/downloads/SOL/fai/reports/fujitsu_personal-data-in-the-cloud.pdf); letzter Zugriff: 09.07.2013, 11:35 Uhr
- [Goo11] GOOGLE INC.: *Google APIs Terms of Service*. Online-Quelle, 2011. – <https://developers.google.com/terms>; letzter Zugriff: 20.07.2013, 23:54 Uhr
- [Ham09] HAMMER, Eran: *OAuth Security Advisory: 2009.1*. Online-Quelle, 2009. – <http://oauth.net/advisories/2009-1>; letzter Zugriff: 19.06.2013, 16:21 Uhr
- [Ham10a] HAMMER, Eran: *Introducing OAuth 2.0*. Online-Quelle, 2010. – <http://hueniverse.com/2010/05/introducing-oauth-2-0/>; letzter Zugriff: 15.06.2013, 21:42 Uhr



- [Ham10b] HAMMER, Eran: *The OAuth 1.0 Protocol*. Online-Quelle, 2010. – <http://tools.ietf.org/html/rfc5849>; letzter Zugriff: 28.07.2013, 22:49 Uhr
- [Ham11a] HAMMER, Eran: *The OAuth 1.0 Guide*. Online-Quelle, 2011. – <http://www.hueniverse.com/oauth>; letzter Zugriff: 12.06.2013, 22:50 Uhr
- [Ham11b] HAMMER, Eran: *The OAuth 1.0 Guide - Protocol Workflow*. Online-Quelle, 2011. – <http://hueniverse.com/oauth/guide/workflow>; letzter Zugriff: 18.06.2013, 23:21 Uhr
- [Ham12] HAMMER, Eran: *OAuth 2.0 and the Road to Hell*. Online-Quelle, 2012. – <http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell>; letzter Zugriff: 15.06.2013, 21:47 Uhr
- [Har12] HARDT, Dick: *The OAuth 2.0 Authorization Framework*. Online-Quelle, 2012. – <http://tools.ietf.org/html/rfc6749>; letzter Zugriff: 15.06.2013, 21:33 Uhr
- [Her03] HERTZOG, Ute: *Solaris 9 Systemadministration*. 1. Auflage. Markt+Technik, 2003. – ISBN 978-3827266187
- [Hö11] HÖLLWARTH, Tobias: *Cloud Migration*. 1. Auflage 2011. mitp, 2011. – ISBN 978-3-8266-9177-5
- [Lin02] LINGMANN, Thomas: *Datenverschlüsselung*. C&L Computer- und Literaturverlag, 2002. – ISBN 3-932311-87-8
- [Man08] MANDL, Peter: *Grundkurs Betriebssysteme*. 1. Auflage 2008. Vieweg + Sohn Verlag, GWV Fachverlage GmbH, 2008. – ISBN 978-3-8348-0392-4
- [Man13] MANDL, Peter: *Grundkurs Betriebssysteme*. 3., aktualisierte und erweiterte Auflage. Springer Vieweg, 2013. – ISBN 978-3-8348-2301-4
- [Mic12] MICROSOFT CORPORATION: *Vertrag über Microsoft-Dienste*. Online-Quelle, 2012. – <http://windows.microsoft.com/de-de/windows-live/microsoft-services-agreement>; letzter Zugriff: 20.07.2013, 19:54 Uhr
- [Mic13] MICROSOFT CORPORATION: *Microsoft Developer Services Agreement*. Online-Quelle, 2013. – <http://msdn.microsoft.com/en-US/live/cc300389>; letzter Zugriff: 11.08.2013, 16:44 Uhr
- [NM95] NEBEL, Ernesto ; MASINTER, Larry: *Form-based File Upload in HTML*. Online-Quelle, 1995. – <http://www.ietf.org/rfc/rfc1867.txt>; letzter Zugriff: 17.07.2013, 13:07 Uhr
- [OAu07] OAUTH CORE WORKGROUP: *OAuth Core 1.0*. Online-Quelle, 2007. – <http://oauth.net/core/1.0/#anchor9>; letzter Zugriff: 08.08.2013, 00:35 Uhr
- [OAu09] OAUTH CORE WORKGROUP: *OAuth Core 1.0 Revision A*. Online-Quelle, 2009. – <http://oauth.net/core/1.0a/>; letzter Zugriff: 19.06.2013, 16:24 Uhr
- [Pie13] PIERRE AUDOIN CONSULTANTS: *Der Ausbau von Private CloudStrukturen in deutschen Unternehmen nimmt Fahrt auf*. Online-Quelle, 2013. – <http://www.pironet-ndh.com/site/pndh-website-site/resource/content/>

- itko/pac-private-cloud-studie/PAC\_Private-Cloud-Studie\_2013.pdf; letzter Zugriff: 09.07.2013, 11:50 Uhr
- [Riv92] RIVEST, Ron: *The MD5 Message-Digest Algorithm*. Online-Quelle, 1992. – <http://www.ietf.org/rfc/rfc1321.txt>; letzter Zugriff: 01.08.2013, 16:54 Uhr
- [Sch10] SCHWENK, Jörg: *Sicherheit und Kryptographie im Internet*. 3., überarbeitete Auflage. Vieweg + Teubner Verlag, Springer Fachmedien Wiesbaden, 2010. – ISBN 978-3-8348-0814-1
- [SPS11] SPITZ, Stephan ; PRAMATEFTAKIS, Michael ; SWOBODA, Joachim: *Kryptographie und IT-Sicherheit*. 2., überarbeitete Auflage. Vieweg + Teubner Verlag, Springer Fachmedien Wiesbaden GmbH, 2011. – ISBN 978-3-8348-1487-6
- [The12] THE UNITED STATES - DEPARTMENT OF JUSTICE: *Justice Department Charges Leaders of Megaupload with Widespread Online Copyright Infringement*. Online-Quelle, 2012. – <http://www.justice.gov/opa/pr/2012/January/12-crm-074.html>; letzter Zugriff: 13.06.2013, 19:33 Uhr
- [Web13] WEBER, Mathias: *Bundestagswahl 2013*. Online-Quelle, 2013. – <http://www.bitkom.org/de/themen/75591.aspx>; letzter Zugriff: 13.06.2013, 19:36 Uhr

