Cloud Storage Pool - Virtuelle Vernetzung von Speicherdiensten in der Cloud

eingereicht von
Ing. Peter Völkl, BA MSc

MASTERARBEIT

zur Erlangung des akademischen Grades
Master of Arts in Business
(MA)

an der Ferdinand Porsche FernFH
Studienrichtung Wirtschaftsinformatik

Begutachter: Dipl.-Ing. Dr. Igor Miladinovic

Wien, am 01.06.2014

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Ort, Datum

Unterschrift

Signaturwert	3x0A2Zq5/0MPyk2TEg+f8qYmhpgU6wUXj/7727mUt7aRlDZk/RITlNMfXo70vDxtNUlHKAGU8 to4uIQzDiRoQg==	
SIGNATURINFORMATIONEN	**Unterzeichner**	Peter Völkl
	Aussteller-Zertifikat	CN=a-sign-premium-mobile-03,OU=a-sign-premium-mobile-03,O=A-Trust Ges. f. Sicherheitssysteme im elektr. Datenverkehr GmbH,C=AT
	Serien-Nr.	1151002
	Methode	urn:pdfsigfilter:bka.gv.at:binaer:v1.1.0
	Parameter	etsi-bka-atrust-1.0:ecdsa-sha256:sha256:sha256:sha1
Prüfinformation	Signaturpruefung unter: http://www.signaturpruefung.gv.at	
Hinweis	Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument ist gemäß § 4 Abs. 1 Signaturgesetz einem handschriftlich unterschriebenen Dokument grundsätzlich rechtlich gleichgestellt.	
Datum/Zeit-UTC	2014-06-01T20:36:26Z	

Kurzfassung

Es gibt mittlerweile eine Vielzahl an Cloud Services, die auch im Umfeld des Personal Computing Verwendung finden. Ein inzwischen, auch im direkten Anwenderumfeld, an massiver Bedeutung gewonnener Dienst ist der des Cloud Storages. Diese ermöglichen den Anwendern, ihre Daten online im Cloud Speicher des jeweiligen Anbieters abzulegen und weltweit über das Internet darauf zugreifen. Es ist dabei nicht einfach möglich, mehrere Onlinespeicher unterschiedlicher Anbieter zu einem integrierten System zu kombinieren. So ist es zwar möglich mehrere Clientapplikationen unterschiedlicher Anbieter zu installieren, der Abgleich funktioniert dabei jedoch nur lokal solange der PC in Betrieb ist. Eine einheitliche Verschlüsselungslösung, die mit allen Clients gleichermaßen zusammenarbeitet existiert ebenfalls nicht. Im Rahmen dieser Arbeit wurde eine Zwischenschicht zwischen dem Anwender und seinen Applikationen und den einzelnen Cloud Storage Services entwickelt, welche einen einheitlichen, integrierten, Zugriff auf alle angebundenen Datenspeicher ermöglicht. Der Zugriff erfolgt dabei über eine virtuelle Ordnerstruktur, die die einzelnen Speicherdienste integriert. Zusätzlich ist es möglich einzelne Datenspeicherorte zu Verschlüsseln und miteinander zu synchronisieren. Diese Entwicklung ermöglicht dem Anwender eine Betriebssystemunabhängige Verwendung und Kombination von beliebigen Datenspeicherdiensten. Der in dieser Arbeit entwickelte Prototyp ist für die Cloud Speicher Services Dropbox und Google Drive ausprogrammiert und ermöglicht eine einfache Erweiterung um beliebig viele weitere Serviceanbieter. Der Zugriff durch den Anwender erfolgt dabei über eine Weboberfläche und eine direkte Anbindung von Anwendungen über eine WebDAV Schnittstelle.

Schlagwörter: Cloud Storage, Dropbox, Google Drive, Microsoft Skydrive, Microsoft Onedrive, WebDAV, SaaS, PaaS, IaaS, Cloud Storage Pool, CSP

Abstract

There is a vast number of cloud services, used in the field of personal computing, nowadays. Cloud storage is a service which became more and more important for home users recently. With the usage of cloud storage, users are enabled to store their files online and access them worldwide via internet. It is a big challenge to combine multiple online storages to one integrated system: It is possible to install multiple separate client-software. The synchronization between the local and the online storage works as long as the client PC is operating and switched on. An encryption method which works across all client PCs does also not exist. In the scope of this master thesis, a layer between the user, his storage applications and the online storage services was developed which provides unified and integrated access to the multiple online storages. The access is based on a virtual folder structure which holds links to the particular online storage services. In addition to this, encryption and synchronization between the online storage services has been implemented. The newly developed web-application enables the user for a cross-platform usage and combination of his online storage services. The prototype application implements the Dropbox and Google Drive Services and can be extended to support other service providers. The cloud storage application can be accessed via web-interface through a browser and a direct connection of storage services via WebDav interface.

Keywords: Cloud Storage, Dropbox, Google Drive, Microsoft Skydrive, Microsoft Onedrive, WebDAV, SaaS, PaaS, IaaS, Cloud Storage Pool, CSP

Danksagung

Ich möchte mich bei allen bedanken, die mich in meinen Studien und bei der Entstehung dieser Arbeit unterstützt haben.

Mein besonderer Dank gebührt meiner Frau Anna, die mich in meinen beiden Masterstudien immer unterstützt hat und mir immer half, die nötige Kraft und Ausdauer dafür zu finden.

Auch möchte ich mich recht herzlich bei Herrn Mag. Dr. Oliver Jorns und Herrn Dipl. Inform. MSc. Telecoms Joachim Zeiß bedanken, mein Interesse für das Thema geweckt und mich zur weiteren Ausführung in Form meiner Master These motiviert haben. Weiter möchte ich mich in diesem Zusammenhang bei Dipl.-Ing. Dr. Igor Miladinovic für die Übernahme der Betreuung meines Vorhabens bedanken.

Die beste Idee nützt niemandem, wenn sie nicht verwertet wird.
(Walter Alteiby)

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Es gibt mittlerweile eine Vielzahl an Cloud Services, die auch im Umfeld des Personal Computing Verwendung finden. Ein an massiver Bedeutung gewonnener Dienst ist der des Cloud Storages. Dabei können Anwender ihre Daten online im Cloud Speicher des jeweiligen Anbieters ablegen und weltweit über das Internet darauf zugreifen. Die Zugriffe erfolgen dabei meist entweder direkt über eine Weboberfläche oder über einen PC Client, der die Daten mit dem lokalen Dateisystem des PCs synchronisiert und damit auch offline verfügbar hält. Der Anwender muss sich hier nicht mehr selbst um den Datenabgleich mit dem Onlinespeicher kümmern, sondern speichert die Daten lediglich in der lokal überwachten Dateistruktur ab. Der Client kümmert sich automatisch um den nötigen Abgleich mit den Onlinedaten.

Es ist dabei nicht einfach möglich, mehrere Onlinespeicher unterschiedlicher Anbieter zu einem integrierten System zu kombinieren. So ist es zwar möglich mehrere Clientapplikationen unterschiedlicher Anbieter zu installieren, der Abgleich funktioniert jedoch nur lokal solange der PC in Betrieb ist. Eine einheitliche Verschlüsselungslösung, die mit allen Clients gleichermaßen zusammenarbeitet existiert ebenfalls nicht.

Durch die Nutzung von Cloud Storage Services ergeben sich aber auch einige Probleme, die es zu beachten und zu lösen gilt. So spielen die Datensicherheit und der Datenschutz in vielen Bereichen eine wesentliche Rolle. Werden Daten nur bei einem einzelnen Anbieter abgespeichert und gehen verloren, oder werden sie durch einen Sicherheitsvorfall für mögliche Angreifer zugreifbar, ist die Sicherheit der Daten gefährdet.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Recherche nach Lösungsmöglichkeiten die Probleme der verteilten Speicherung und der Verschlüsselung unterschiedlicher Cloud Storage Anbieter zu lösen, indem eine eigene Onlineapplikation als Zwischenschicht geschaffen wird. Diese soll direkt über eine Weboberfläche erreichbar sein und die Daten über eine einheitliche Schnittstelle für den Client PC zugänglich machen. Die Anwendung soll als Prototyp für die beiden am häufigsten genutzten Cloud Speicherdienste implementiert werden. Das Anwendungsdesign soll modular erweiterbar gestaltet werden, damit jederzeit zusätzliche Funktionen ergänzt werden können. Auch soll die Installation auf eigenen Systemen einfach und betriebssystemunabhängig möglich sein. Als Arbeitstitel dieser zu entwickelnden Zwischenschicht wird der Name Cloud Storage Pool (CSP) gewählt.

Die damit verbundene Forschungsfrage und These lauten: **Es ist möglich, eine betriebssystemunabhängige Webanwendung zu entwickeln, die als Zwischenschicht zwischen Anwender und mehreren Cloud Storage Services arbeitet und die redundante und verschlüsselte Speicherung von in der Cloud gespeicherten Daten erlaubt, so dass der Anwender über eine virtuelle Dateistruktur direkt, und ohne der Notwendigkeit des Wissens über den dahinterliegenden Speicherort, darauf zugreifen kann.**

Der Aufbau dieser grundlegenden Funktion ist in Abbildung 1 skizziert. Der Anwender meldet sich mit seinem PC direkt beim Cloud Storage Pool an und bekommt danach direkten Zugriff auf die virtuelle Dateistruktur. Die Ablage und Verteilung der Daten in den Cloud Storages erfolgt automatisch im Hintergrund.

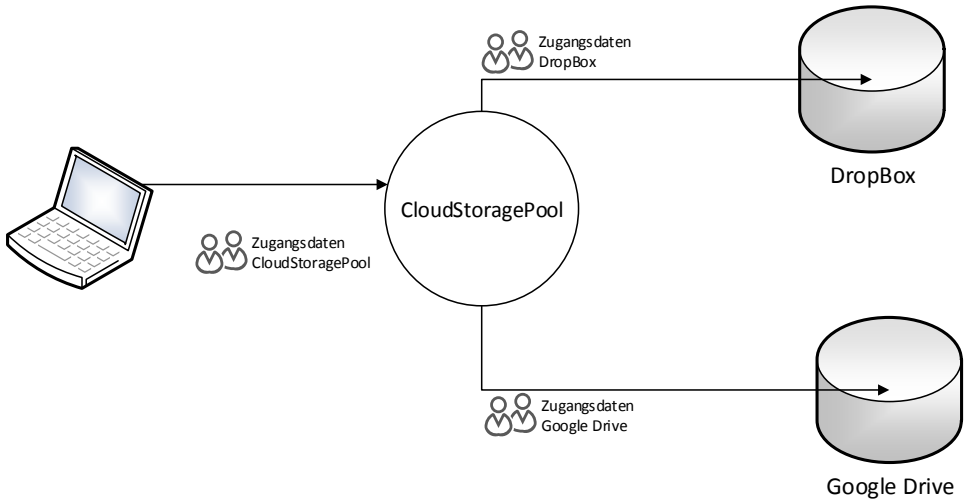

Abbildung 1 - Vernetzung mehrerer Cloud Storage Anbieter durch den Cloud Storage Pool

1.3 Aufbau der Arbeit

Die Arbeit beschäftigt sich zum Beginn mit dem Entstehen und den Grundlagen des Cloudcomputing und der Cloud Storage Services, sowie einem Überblick zum Thema Cloud Storage Security. Danach wird der aktuelle Stand der Technik der Cloud Storages und der bei PC Anwendern am häufigsten genutzten Cloud Storage Services zusammengefasst.

Der Hauptteil der Arbeit beschäftigt sich dann mit dem Design und der Implementierung der Applikation Cloud Storage Pool, der damit verbundenen Bewertung der Forschungsfrage und These, sowie dem weiteren Ausblick auf sich daraus ergebende zukünftige Weiterentwicklungen.

2 Cloudcomputing

2.1 Entstehung und Definition

Cloud Computing ist die mittlerweile fünfte Generation des Computing. Der Begriff des Computing ist ein Synonym für „counting and calculating“ und definiert die systematische Analyse von Algorithmen und Prozessen, die für die Beschreibung und Abarbeitung von Information zuständig sind. [Pad12, S. 182]

Am Beginn der Entstehung des Cloudcomputings stand das Mainframe Computing. Dabei handelt es sich anfangs um die zentrale Abarbeitung von Batchaufträgen, deren Befehlsabfolgen meist mittels Lochkarten bereitgestellt wurden. Die Interaktion mit den Computern war so meist nur Programmierern möglich. Später wurde das Mainframe Computing um die Möglichkeit von Terminalanbindungen erweitert, mit deren Hilfe Befehle an den Mainframe gesendet werden und Ausgaben entgegengenommen werden konnten. [Pad12, S. 183f]

Die nächste Generation war das Personal Computing (PC). Dabei wurden kleine, eigenständige und - im Vergleich zum Mainframe Computing - wesentlich günstigere Computing Systeme geschaffen, welche ein eigenes Betriebssystem hatten und eine lokale und dezentrale Abarbeitung von Anwendungen ermöglichten. Das Personal Computing wurde schnell durch das Network Computing ergänzt und dann größtenteils abgelöst. Beim Network Computing handelt es sich um einen Verbund mehrerer Personal Computing Systeme mittels Computernetzwerk. Die Architektur wird hierbei meist über eine Client-Server Kommunikation abgebildet. Dabei handelt es sich um PCs mit unterschiedlichen Rollen. Der Client ist für die Benutzerinteraktion und die lokale Abarbeitung von Anwendungen zuständig und der Server für die zentrale Abarbeitung von gemeinsamen oder sehr komplexen Anwendungen. Zusätzlich zu Client-Server Systemen wurden Peer-to-Peer Systeme ermöglicht, welche ebenfalls direkte Verbindungen zwischen mehreren Clients erlauben. Dadurch können diese, ohne einen zentralen Server als Verteilerknoten, direkt Daten austauschen und gemeinsame Berechnungen durchführen. [Pad12, S. 184ff]

Das Network Computing wurde später um das Internet Computing erweitert. Dabei wurde weltweit eine Vielzahl von Computernetzwerken zu einem gemeinsamen Netzwerk verbunden, dem World Wide Web (WWW) oder Internet. Damit wurde die Möglichkeit geschaffen, mehrere Client-Server und Peer-to-Peer Anwendungen über alle angebundenen Netzwerke verteilt zu nutzen. Um eine einheitliche Datenkommunikation zu ermöglichen, wurden zu diesem Zweck mehrere Protokolle definiert. Beispiele sind das Hyper Text Transfer Protokoll (HTTP) [RFC99], das Simple Mail Transport Protokoll (SMTP) [RFC08] und die Voice over Internet Protokolle (VoIP), wie H.323 [Int09] und das

Session Initiation Protocol (SIP) [RFC02]. Um einzelne Server und Services im Internet eindeutig adressieren zu können, dient der Uniform Resource Locator (URL). Er definiert sowohl das zu verwendende Protokoll, als auch die Erreichbarkeit des Servers über seine Netzwerkadresse oder seinen Namen und den Aufruf des darauf befindlichen Services. Für die Adressierung der einzelnen Teilnehmer im Internet werden IP-Adressen verwendet, welche wiederum über das Domain Name System (DNS) mit, für den Anwender leichter zu erfassenden, Namen versehen werden können. Dabei werden die Relationen zwischen Name und Netzwerkadresse von speziellen DNS-Servern verwaltet. Möchte ein Computer auf einen mittels DNS adressierten Service zugreifen, fragt er bei seinem zuständigen DNS-Server nach und bekommt die zugehörige IP-Adresse, mit deren Hilfe er dann die direkte Verbindung herstellen kann. [Pad12, S. 188f]

Der unmittelbare Vorgänger des Cloud Computing ist das Grid Computing. Es verfolgt die Strategie der verteilten Ressourcennutzung über mehrere Server. Dabei werden die einzelnen Ressourcen, wie Rechenleistung, Datenmanagement, Anwendungen, Koordination und Wissen, als eigene Services betrachtet, welche über ein entsprechendes Grid Management zusammengefasst und nutzbar gemacht werden. Für die Verteilung der Ressourcen ist der Grid Ressource Broker zuständig. Dieser kümmert sich um die Verteilung der Anfragen gemäß ihrer Priorisierung und Herkunft auf unterschiedliche gleichwertige Ressourcen vor. Das Grid Computing ermöglicht damit in vielen Fällen eine effizientere und verfügbarkeitssicherere Abarbeitung komplexer Anwendungen. Die im Rahmen des Grid Computing entwickelten Vorgehensweisen und Ressourcenschichten stellten die Basis des darauf aufbauenden Cloud Computing dar. Im Gegensatz zum Cloud Computing werden hier die Aufgaben nicht durch einen einzelnen Anbieter verteilt und gesteuert. Die Ressourcennutzung erfolgt im Cloudcomputing immer über eine Vereinbarung zwischen Anbieter und Anwender, beim Grid Computing werden dagegen die verfügbaren Ressourcen gemäß der Priorisierungsentscheidungen des Grid Ressource Brokers zugeteilt. Die Teilnehmer stellen beim Grid Computing auch meist selbst Ressourcen zur Verfügung, welche wiederum von den anderen Teilnehmern mitgenutzt werden können. [Pad12, S. 189ff]

Der Begriff Cloud-Computing bezeichnet das Auslagern von einzelnen IT-Services hin zu Anbietern dieser Services im Internet. Die Möglichkeiten sind hierbei sehr vielfältig und beschränken sich nicht allein auf die Zurverfügungstellung von Online-Speicherplatz, sondern beziehen sich auf die gesamte Bandbreite vom einfachen Webservice bis hin zu kompletter virtueller Infrastruktur. Ein Vorteil liegt hierbei in der einfachen Skalierbarkeit der in Anspruch genommenen Services, da sich meist viele Anwender die Ressourcen des Anbieters, je nach gewünschtem Bedarf, teilen. Die Einstiegskosten sind beim Cloud Computing entsprechend geringer als bei der Investition in eigene Ressourcen. Der Hauptvorteil liegt daher in den sich daraus ergebenden wirtschaftlichen Vorteilen durch die variable Mitnutzung komplexer Ressourcen. Beim Cloud Computing handelt es sich, im

Gegensatz zum Grid Computing, um das Outsourcing von IT-Services und nicht um eine Vernetzung bei der die gemeinsame Ressourcennutzung im Vordergrund steht.

Die Evolution des Computing ist in Abbildung 2 in ihren Zusammenhängen skizziert.

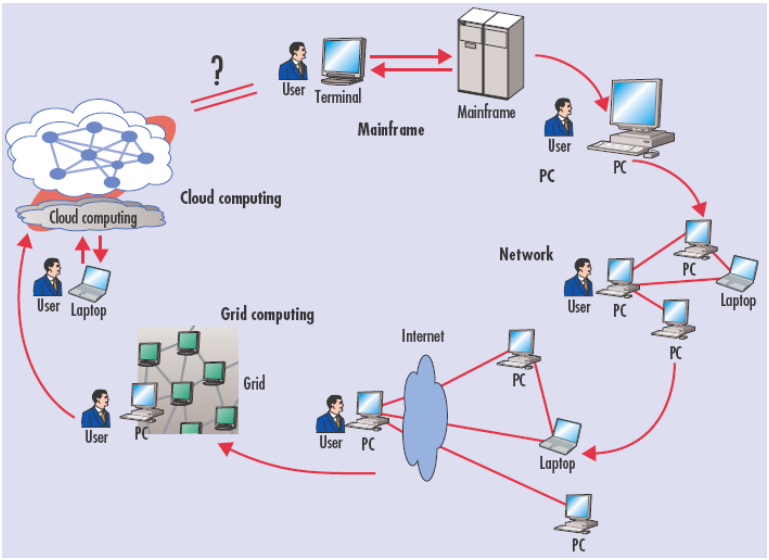

Abbildung 2 - Evolution des Computing [JuW11, S. 1766]

Die Definition der US-amerikanischen Standardisierungsstelle NIST (National Institute of Standards and Technologie) für Cloudcomputing lautet [Nat11, S. 2]:
Cloud Computing ist ein Modell, das es erlaubt bei Bedarf, jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können.

Die NIST definiert dabei fünf Charakteristiken eines Cloud Services [Nat11, S. 2] [Gug13, S. 11]:

1. On-demand Self Service: Automatische Zurverfügungstellung der Ressourcen
2. Broad Network Access: Die Services sind über das Netz verfügbar
3. Resource Pooling: Die Ressourcen werden den Anwender über einen gemeinsamen Pool zur Verfügung gestellt.
4. Rapid Elasticity: Zusätzliche Ressourcen werden schnell und elastisch zur Verfügung gestellt. Eine Entscheidung des Anwenders für mehr Ressourcen ist schnell und einfach möglich.
5. Measured Services: Die Nutzung der Ressourcen wird genau dokumentiert und ist für den Anwender abrufbar.

Weiter werden vier Bereitstellungsmodelle unterschieden [Nat11, S. 3] [Bau11, S. 27f]:

1. Private Cloud: Die Cloud Services werden nur für eine einzelne Institution betrieben.
2. Public Cloud: Die Cloud Services werden für die Allgemeinheit oder eine große Anwendergruppe betrieben, welche nicht auf einzelne Institutionen beschränkt sind.
3. Community Cloud: Die Cloud Services werden von mehreren Institutionen geteilt.
4. Hybrid Cloud: Für sich eigenständige Services werden über standardisierte Schnittstellen gemeinsam genutzt.

Die Unterschiede der einzelnen Bereitstellungsmodelle sind in Abbildung 3 schematisch dargestellt. Hier nutzen Benutzer A und Anbieter X eine Hybride Cloud, wodurch Services einer Public Cloud und einer Private Cloud vernetzt wurden. Benutzer B und C nutzen jeweils ihre eigene Private Cloud und Anbieter Y stellt Services einer Public Cloud zur Verfügung. Würden die beiden Benutzer B und C Teile ihrer Private Cloud Services gemeinsam nutzen, würde es sich um eine Community Cloud handeln. Anbieter Z bietet dahingegen selbst mehrere Public Cloud Services an.

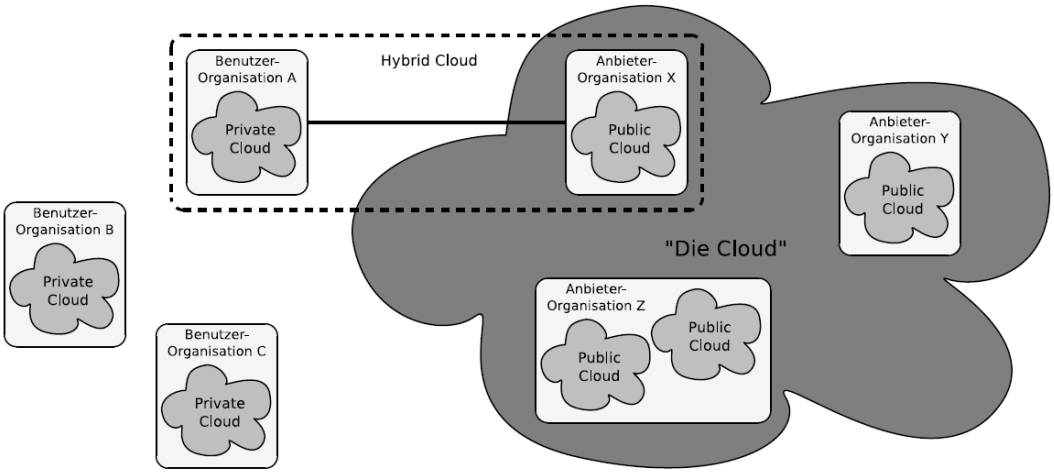

Abbildung 3 - Public Cloud, Private Cloud, Hybrid Cloud [Bau11, S. 28]

2.2 Services in der Cloud

Innerhalb der Cloudservices wird zwischen vier verschiedenen Servicemodellen unterschieden: Humans as a Service (HuaaS), Software as a Service (SaaS), Platform as a Service (PaaS) und Infrastructure as a Service (IaaS). Die Architektur kann in Form eines Schichtenmodells beschrieben werden, da die einzelnen Servicearten aufeinander aufbauen und die jeweils unten liegende Serviceschicht benötigen. Das Schichtenmodell der Cloud-Architektur ist in Abbildung 4 inklusive der in den jeweiligen Schichten befindlichen Services dargestellt.

Abbildung 4 - Cloud-Architektur Schichtenmodell [Bau11, S. 30]

Die vier Servicetypen, beziehungsweise Schichten, werden nachfolgend kurz beschrieben.

2.2.1 Humans as a Service (HuaaS)

Die oberste Schicht bezieht sich auf den menschlichen Faktor im Cloud-Computing und setzt auf den übrigen Serviceschichten auf. Sie bindet die Dienstleistung der Ressource Mensch in die Betrachtung der möglichen Services mit ein. Diese übergeordnete Ressource kommt überall dort zum Einsatz, wo die menschlichen Fähigkeiten denen einer reinen computerbasierten Lösung überlegen sind. Das sind beispielsweise Übersetzungsdienste oder Design-Dienste, welche ein hohes Maß an Kreativität benötigen. Eine Unterkategorie stellt das Crowdsourcing dar, bei dem eine Gruppe menschlicher Ressourcen Aufgaben unterschiedlicher Komplexität und Umfang übernimmt. [Bau11, S 39f]

2.2.2 Software as a Service (SaaS)

Der SaaS Layer beinhaltet alle Arten von Softwareanwendungen, welche direkt durch die jeweiligen Anwender, ohne der Verwendung einer zusätzlichen Softwareinstallation, verwendet werden können. Im Hintergrund werden zusätzlich die für die Anwendung benötigten Ressourcen in der Cloud zur Verfügung gestellt. Diese werden dann über die darunter liegenden Layer PaaS und IaaS abgebildet. Es gibt unterschiedliche Ausprägungen der SaaS Services. Diese beginnen bei einem einfachen Webdienst, welcher über eine Internetseite erreichbar ist, wie Geosuchen, und reichen bis hin zu vollen Anwendungen, wie Office Applikationen. [Bau11, S. 37]

Einige Beispiele für SaaS Services sind in Tabelle 1 aufgelistet.

Tabelle 1 - SaaS Angebote und Werkzeuge [Bau11, S. 38] (Namen aktualisiert)

Organisation, Cloud Dienst	Beschreibung, Referenz
Adobe Photoshop Express	Online Bildbearbeitung [Ado14]
fluidOps eCloudManager SAP Edition	SAP Landscape as a Service [flu14]
Google Docs	Online Office Anwendungen [Goo14]
Google Maps API	Dienst zur Integration von Landkarten und geographischen Informationen [Goo141]
Google OpenSocial	Übergreifende Programmierschnittstelle zur Integration sozialer Netze in Anwendungen [Goo142]
OpenID Foundation, OpenID	Verteiltes System zur Verwaltung systemübergreifender Benutzeridentitäten [Ope14]
Microsoft Office Online	Online Office Anwendungen [Mic141]
Salesforce, Salesforce.com	Erweiterbares CRM-System [sal14]

2.2.3 Platform as a Service (PaaS)

Bei den PaaS Services handelt es sich um eine Schicht zwischen den SaaS und den IaaS. Hier werden einerseits die zugrundeliegenden Ressourcen des IaaS (siehe 2.2.4) genutzt um andererseits Software des SaaS Layers implementieren zu können. Die Zielgruppe dieser Entwicklungs- und Konfigurationsplattformen sind hauptsächlich Entwickler und Systemarchitekten, welche direkt an der Entwicklung von SaaS Lösungen beteiligt sind. [Bau11, S. 35]

Einige Beispiele für PaaS Services sind in Tabelle 2 aufgelistet.

Tabelle 2 - PaaS Angebote und Werkzeuge [Bau11, S. 36] (Namen aktualisiert)

Organisation, Cloud Dienst	Beschreibung, Referenz
Akamai EdgePlatform	Content, Site, Application Delivery [Aka14]
Facebook Platform	Umgebung für Anwendungen im sozialen Netzwerk Facebook [Fac14]
Google App Engine	Skalierbare Ausführungsumgebung für Web-Anwendungen [Goo143]
Microsoft Azure	Entwicklungs- und Ausführungsumgebung für Windows Anwendungen [Mic142]
Microsoft Windows OneDrive	Plattform zum Datenabgleich zwischen heterogenen Endgeräten [Mic143]
NetSuite SuiteFlex	Werkzeug zur Geschäftsprozessentwicklung in NetSuite [Net14]
Salesforce Force.com	Entwicklung und Betrieb von verteilten Web Anwendungen [sal141]
Zoho Creator	Entwicklung und Betrieb von Datenbank-basierten Web-Anwendungen [Zoh14]

2.2.4 Infrastructure as a Service (IaaS)

Bei der Bereitstellung von IaaS Services handelt es sich um die grundlegendste Schicht der Cloudservices. Hier werden den Anwendern oder darüber liegenden Services, grundlegende Infrastrukturteile zur Verwendung angeboten. Es handelt sich dabei jedoch um eine stark abstrahierte Sicht auf die Ressourcen, da die tatsächliche Ressourcenverwendung im Hintergrund für den Anwender, in der Regel, nicht steuerbar ist. Gerade dies ermöglicht es dem Anbieter, den Umfang der angebotenen Ressourcen dynamisch zu skalieren und zu vergeben. Beispiele für IaaS Services sind: Rechenleistung, Speicherplatz und komplette virtuelle Hardwaresysteme, wobei auch diese wiederum dynamisch skalierbar sind. Es ist dabei möglich einzelne virtuelle Ressourcen in tatsächliche physische Konfigurationen zu integrieren, oder komplette IT-Systeme rein aus virtuellen Ressourcen zusammenzustellen.

Das in dieser Arbeit wesentliche Beispiel für die Integration virtueller Ressourcen in physische Ressourcen sind die Cloudstorage-Services, welche dem Anwender eine leicht skalierbare Dateiablage in der Cloud ermöglichen, die sich jedoch nahtlos in das lokale Dateisystem einbinden lässt. [Bau11, S. 31f]

Einige Beispiele für SaaS Services sind in Tabelle 3 aufgelistet.

Tabelle 3 - IaaS Services, Angebote und Werkzeuge [Bau11, S. 33f] (gekürzt und Namen aktualisiert)

Organisation, Cloud Dienst	Beschreibung, Referenz
Amazon Elastic Compute Cloud (EC2)	Virtuelle Server [Ama14]
Amazon Simple Storage Services (S3)	Massenspeicher [Ama141]
Amazon SimpleDB	Datenbank as a Service (DaaS) [Ama142]
BlueLock Virtual Cloud Computing	Virtuelle Server [Blu14]
BlueLock Virtual Recovery	Wiederherstellung virtueller Server bei Störungen [Blu14]
Dropbox Cloud Storage	Massenspeicher [Dro14]
GoGrid Cloud Hosting	Virtuelle Server [GoG14]
GoGrid Cloud Storage	Massenspeicher [GoG14]
Google Big Table	Virtueller Speicher für strukturierte Daten [Cha06]
Google File System	Verteiltes Dateisystem [Ghe03]
Rackspace Cloud Servers	Vorkonfigurierte virtuelle Server [Rac14]
Skytap Virtual Lab	Hybrid Cloud Testumgebungen [Sky14]

2.3 Cloud Storage Services

Analog zu der Definition von Cloud Services in die Kategorien private, public und hybrid, können auch die Cloud Storage Services in private Cloud Storage, public Cloud Storage und hybrid Cloud Storage eingeteilt werden. Die Definition ist hierbei dieselbe und wurde bereits in Kapitel 2.1 genauer beschrieben.

Cloud Speicherdienste können zum Teil sowohl als IaaS (siehe 2.2.4), als auch als PaaS (siehe 2.2.3) Services eingestuft werden, da sie oft auch als Plattform für darauf aufbauende Cloud Software geführt werden. In der Regel handelt es sich dabei aber grundlegend um Services der Kategorie IaaS, in Form von reinen Speicherressourcen inklusive ihrer Verwaltung und möglichen, darauf aufbauenden PaaS und SaaS (siehe 2.2.4) Anwendungen. Oft wird dabei auch von Storage as a Service gesprochen.

Der direkte Vorgänger der Cloud Storage Technologien waren die Netzwerkspeicher. Verbreitete Beispiele hierfür sind das von Sun Microsystems entwickelte Network File System (NFS) [RFC03], welches hauptsächlich in Unix und Linux Umgebungen zum Einsatz kommt, und das von Microsoft entwickelte Server Message Block (SMB) [Mic145]

Protokoll und seinem Nachfolger dem Common Internet File System (CIFS) [Mic146], welche sich primär in Microsoft Systemen wiederfindet.

Genauso, wie im Bereich der Netzwerkspeicher, wurden für den Zugriff auf Cloud Speicher von den jeweiligen Anbietern eigene Protokolle, wie beispielsweise das Google File System (GFS) [Ghe03] und das Hadoop Distributed File System (HDFS) [Apa14] entwickelt. [JuW11, S. 1766]

In Cloud Storages werden die Daten auf mehrere Serverknoten verteilt gespeichert und ebenfalls verteilt ausgeliefert. Diese Art der Verteilung wird oft auch als Sektor System Architektur bezeichnet. Dadurch kann eine höhere Bandbreite bei der Auslieferung an eine große Zahl von Clients erreicht werden als durch den Zugriff auf einen einzelnen Server möglich wäre. Durch die Verteilung auf mehrere Serverstandorte wird die Last damit auf mehrere Datenleitungen verteilt. Die Verteilung und Spiegelung der abgelegten Daten erfolgt, je nach Anbieter und verwendeter Technologie, unterschiedlich und kann dabei sowohl dateibasiert, als auch blockbasiert erfolgen. Die Verwaltung der Benutzerzugriffe und Security wird dabei von einem Master-Knoten übernommen und an die Slave-Knoten weitergereicht. [JuW11, S. 1767]

Der Aufbau einer Sektor System Architektur ist in Abbildung 5 dargestellt.

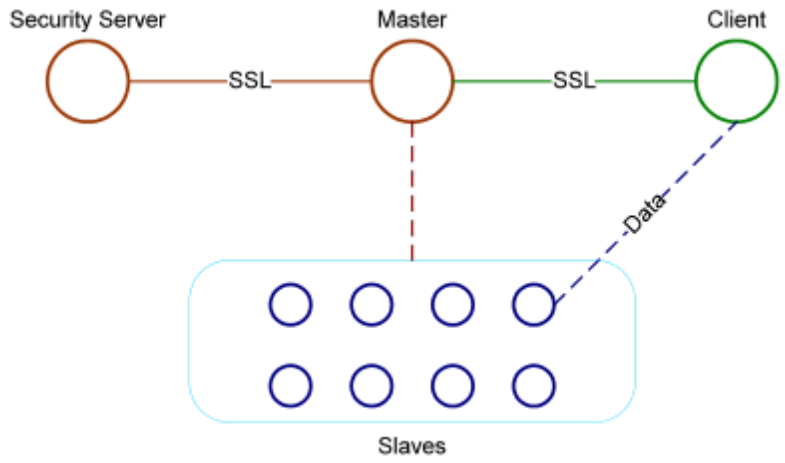

Abbildung 5 - Sektor System Architektur [JuW11, S. 1767]

Durch die bessere Skalierbarkeit und Verteilung von Last und Verfügbarkeit kann durch die Verwendung von Cloud Storage oft eine bessere Kosten- und Performanceeffizienz, als bei herkömmlichen Speicherlösungen erreicht werden. [JuW11, S. 1768]

2.4 Cloud Storage Security

Für die Authentifizierung und die damit verbundene Autorisierung werden bei den einzelnen Cloud Storage Anbietern unterschiedliche Verfahren verwendet. Diese können eigene Implementierungen oder allgemein verfügbare Methoden wie beispielsweise OAuth [Mes14] sein. OAuth ermöglicht die zentrale Verwaltung von Benutzern und Rollen gegenüber Drittanbieter Services. Dabei wird mit Access Token gearbeitet, die vom Autorisierungs-Server an den Client übergeben werden und ihm damit den Zugriff auf die eigentlichen Ressourcen ermöglichen [Bab14, S. 4]. Bei OAuth sind bisher keine Protokollfehler bekannt, welche von Angreifern ausgenutzt werden konnten. Die Hauptangriffsvektoren sind hierbei fehlerhafte Implementierungen auf Seiten des Services oder des Clients, welche zu Schwachstellen führen können [Bab14, S. 13ff].

Auch die Datenübermittlung erfolgt auf unterschiedliche Art und Weise, diese wurden von [Wag14] im Detail gemessen und analysiert. So werden bei Google Drive und Microsoft OneDrive (vormals SkyDrive) die geänderten Daten zur Gänze an den Speicherdienst gesendet und bei Dropbox nur die jeweils geänderten Datenblöcke [Wag14, S. 53 ff]. Dies ist insbesondere von Bedeutung, da mit der Übertragungsmenge und Anzahl der Übertragungen zusammenhängenden Datenteile die Wahl der Verschlüsselungsmethode zusätzlich an Bedeutung gewinnt. Auf diese hat der Anwender jedoch kaum bis überhaupt keinen Einfluss.

Die Daten selbst sind innerhalb der verteilten Speicher des Cloud Storage Anbieters meist über Hashwerte adressiert um sie schnell aufzufinden und Redundanzen innerhalb eines Speicherortes zu vermeiden. Das bedeutet jedoch auch, dass ein möglicher Angriff unter Kenntnis des Hashwertes möglich ist. Hier können fehlerhafte Security-Implementierungen der Anbieter ausgenutzt werden. Einen weiteren möglichen Angriffsvektor stellt in diesem Zusammenhang die Ausnutzung des Geburtstagsparadoxons dar. Hierbei kann dem Speicherdienst der Besitz einer Datei, durch die Generierung einer anderen Datei mit gleichem Hashwert, vorgetäuscht werden (Hash Value Manipulation Angriff). Diese und ähnliche Angriffsszenarien wurden in [Mul11] analysiert und anhand des Cloud Storage Services Dropbox getestet und als durchführbar eingestuft.

Neben der besseren Absicherung der Datenübertragungen und der Verifikation des Dateibesitzes [Mul11, S. 9], könnten die Daten mittels Kryptographie auf Dateiebene, vor und nach ihrer Übertragung an den Cloud Speicher Service abgesichert werden. Dabei können die schwachen Sicherheitsmechanismen der Anbieter, im schlimmsten Fall, lediglich zum Abruf von verschlüsselten Daten ausgenutzt werden.

Ein weiterer wichtiger Aspekt der Nutzung von Cloud Speicherdiensten ist der Speicherort. So ist durch die EU Datenschutzrichtlinie 95/46/EG [Eur00], welche in Österreich durch

das Datenschutzgesetz 2000 (DSG2000) umgesetzt wurde, verboten, personenbezogene Daten aus EU Mitgliedsstaaten an Staaten weiterzugeben, die über keinen gleichwertigen Datenschutz verfügen.

Einige Staaten haben Gesetze erlassen, welche ihnen, in bestimmten Fällen, Zugriff auf die gespeicherten Daten aller Unternehmen des Landes erlauben. Ein Beispiel für eine solche gesetzliche Befugnis ist der USA Patriot Act. Solche Eingriffe werden natürlich in wenigen Unternehmen gerne gesehen, und verstoßen mitunter gegen Datenschutzbestimmungen in anderen Ländern. [Arm09, S. 15]

Im Jahr 2000 wurde von der europäischen Kommission daher zusätzlich das Safe-Harbor-Abkommen verabschiedet, welches gemeinsam mit dem US Handelsministerium [USH14] ausgearbeitet wurde. Es ermöglicht einen Datenaustausch zu US Gesellschaften, sofern diese die Safe-Harbor-Vereinbarung unterzeichnet haben und damit dem Safe-Harbor-Programm beigetreten sind. Dies soll einen ausreichenden Schutz der personenbezogenen Daten gewährleisten, obwohl der grundlegende Datenschutzstandard der USA deutlich unter dem der EU Staaten liegt. [Eur001]

Zur Unterstützung bei der Vertragsgestaltung mit Cloud Service Anbietern wurde von EuroCloud.Austria, gemeinsam mit der Wirtschaftskammer Wien, dem Austrian Standards Institute und dem IT-Cluster der Wirtschaftsagentur Wien ein Katalog mit empfohlenen Bestandteilen von Cloud-Verträgen [Eur12] erstellt, welcher ebenfalls die zu beachtenden Regelungen des Bereichs Cloud Security mit abdeckt.

Auch die Verfügbarkeit und Integrität der abgelegten Daten ist ein wesentlicher Aspekt der Cloud Security. So sollten kritische und wichtige Daten auf mehrere Speicherorte gespiegelt abgelegt werden, um eine Verfügbarkeit der mit ihnen verbundenen Services zu gewährleisten [Kav13, S. 48]. Im Falle von automatischen Abgleichen muss auch die Datenintegrität gewährleistet werden, indem mögliche Dateikonflikte, bei gleichzeitiger Änderung der Daten auf den einzelnen Spiegelungen, erkannt und entsprechend behandelt werden.

3 Related Work

3.1 Technische Grundlagen

Für die Implementierung von Anbindungen von Cloud Storage Services existieren unterschiedliche Application Programming Interfaces (APIs) und Standards. Allen voran stehen das Representational State Transfer (REST) [Fie00] und das Simple Object Access Protocol (SOAP) [W3C07], welche meist eingesetzt werden um den Zugriff auf die Schnittstellen der Services ermöglichen. Diese beiden Protokolle werden nachfolgend kurz beschrieben.

3.1.1 Simple Object Access Protocol (SOAP)

SOAP basierte anfangs auf dem Hyper Text Transfer Protokoll (HTTP) und wurde 1999 zum ersten Mal veröffentlicht. Seit Version 1.1 wurde SOAP um mehrere weitere Transferprotokolle erweitert und ist damit nicht mehr nur auf HTTP allein beschränkt. [Gus02, Pos. 2855] [W3C07]

In SOAP kommt das Datenformat Extensible Markup Language (XML) [W3C06] zum Einsatz. Es definiert und vereinheitlicht damit die Art des Strukturaufbaues der Datenübertragungen. Zusätzlich definiert es, wie die XML-Nachrichten aufgebaut werden müssen und wie die Kommunikation abläuft und transportiert wird. SOAP ist dabei ein zustandsloses Protokoll und nimmt keine Rücksicht auf die Semantik des Nachrichtenverkehrs. Alle diesbezüglichen Mechanismen müssen in den darüber liegenden Schichten berücksichtigt werden. [Gus02, Pos. 2860]

Jede Datenübertragung wird bei SOAP in einen sogenannten SOAP-Envelope (Container) gepackt, welche einen optionalen SOAP-Header und einen verpflichteten SOAP-Body enthält. Sowohl Header, als auch Body können aus mehreren Unterelementen bestehen, die als Header-Blocks und Body-Blocks bezeichnet werden. Der Aufbau eines SOAP-Envelopes ist in Abbildung 6 schematisch und als Codebeispiel dargestellt. [Gus02, Pos. 2885]

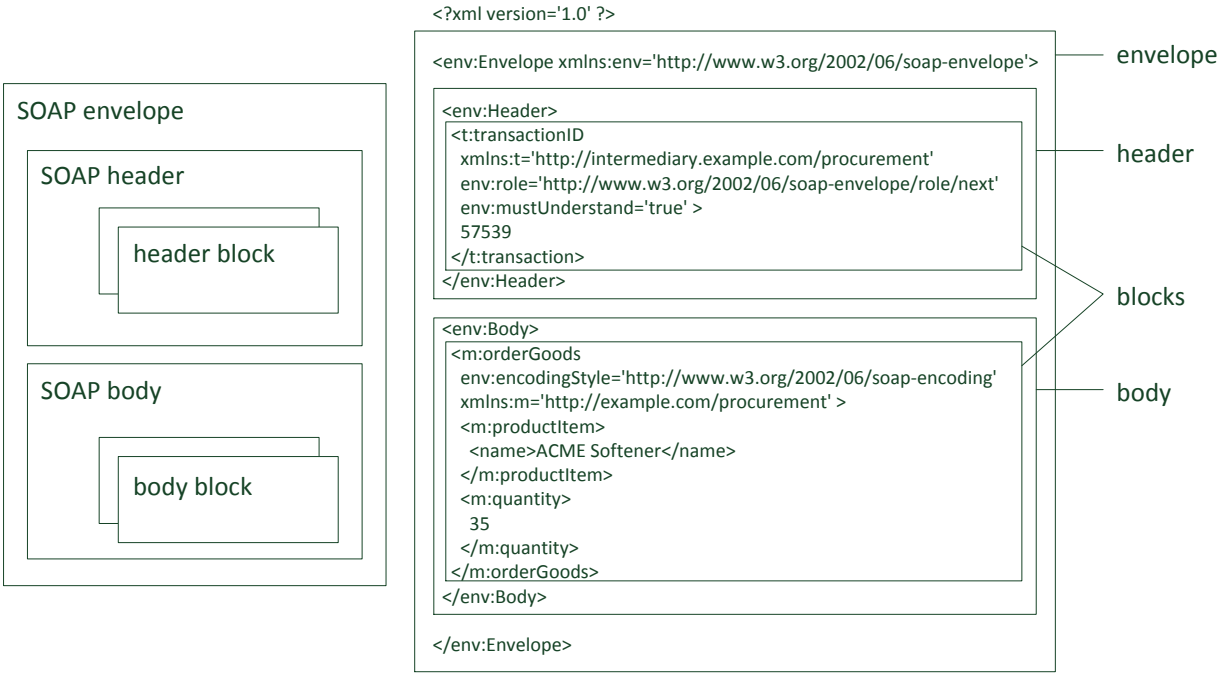

Abbildung 6 - Schema einer SOAP Nachricht [Gus02, Pos. 2885], [Gus02, Pos. 2935]

Der Content innerhalb des SOAP-Bodys unterliegt keiner speziellen Strukturvorgabe. Es gibt jedoch Empfehlungen für dessen Aufbau um zu möglichst vielen Implementierungen kompatibel zu bleiben. Die beiden wichtigsten Definitionen sind der Document-Style und der RPC-Style. Im Document-Style vereinbaren die beiden kommunizierenden Applikationen die Art der Dokumentenstruktur und übertragen dann genau diese Dokumente in Form von SOAP Nachrichten. Die Information über den jeweiligen Dokumententyp wird dabei im Nachrichtenheader übermittelt. Der Empfänger antwortet nach dem Erhalt mit einem Acknowledgement-Document, welches die notwendigen Bestätigungsmeldungen enthält. Im RPC-Style wird zusätzlich zur Dokumentenstruktur des Document-Style der gewünschte Prozeduraufruf eingebettet, mit dem die Daten beim Empfänger verarbeitet werden sollen. So können gezielt unterschiedliche Funktionalitäten des jeweiligen Webservices angesprochen werden. [Gus02, Pos. 2894ff]

Der unterschiedliche Aufbau von Document-Style und RPC-Style wird in Abbildung 7 schemenhaft dargestellt.

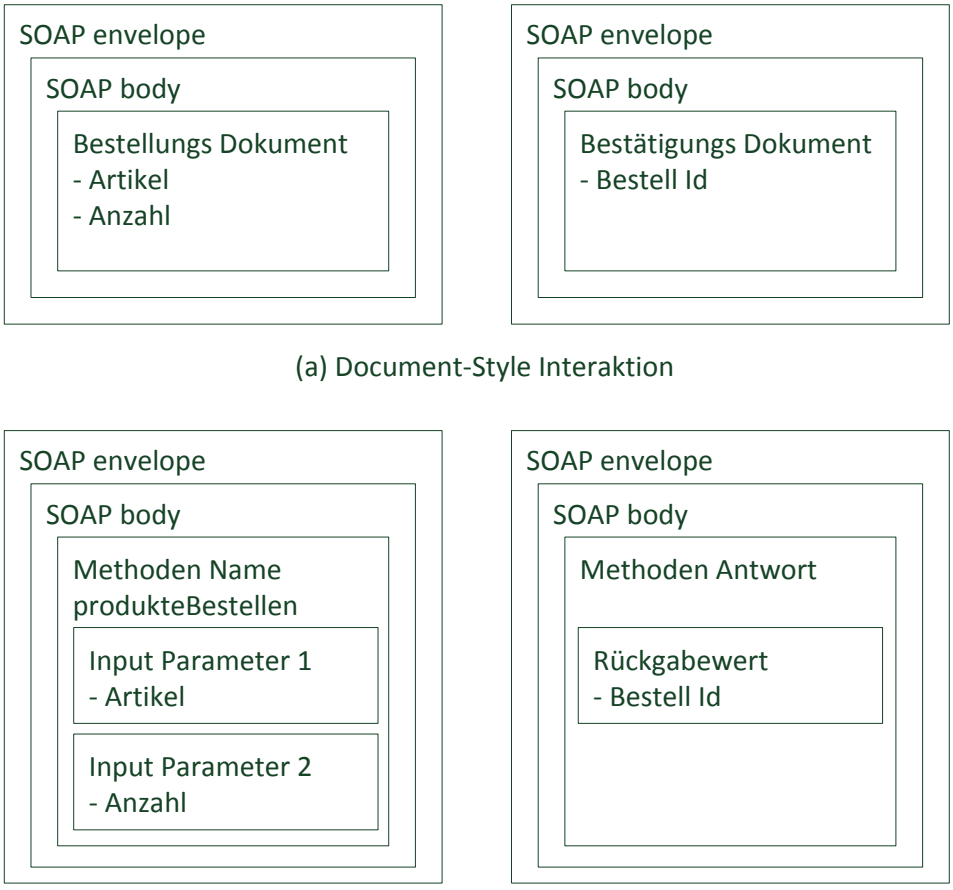

Abbildung 7 - SOAP Nachrichten - Document-Style und RPC-Style [Gus02, Pos. 2922]

Wenn die SOAP Nachrichten mittels HTTP transportiert werden, werden meist die beiden gängigen HTTP-Kommandos GET und POST verwendet. Dabei wird der URL für die Adressierung des Services verwendet und die Übertragung der SOAP-Nachricht erfolgt über den jeweiligen Body des HTTP-Request. [Gus02, Pos. 2966]

3.1.2 Representational State Transfer (REST) und Webbased Distributed Authoring and Versioning (WebDAV)

REST basiert direkt auf dem HTTP Protokoll und nutzt die darin spezifizierten Kommandos und Übertragungsmethoden. Der Architekturstil von REST [Fie00] ist jedoch um eine Stufe abstrakter definiert als HTTP und kann damit auch, ähnlich wie SOAP, mit Hilfe von unterschiedlichen Transportprotokollen eingesetzt werden. [Til11, Pos. 438]

Die fünf Grundprinzipien von REST sind [Til11, Pos. 474]:

- Ressourcen mit eindeutiger Identifikation,

- Verknüpfungen/Hypermedia,
- Standardmethoden,
- unterschiedliche Repräsentationen und
- statuslose Kommunikation.

Die eindeutige Identifikation wird bei REST über HTTP mittels Uniform Resource Identifiers (URIs) sichergestellt. Diese beinhalten die eindeutige Definition des Ziels und Services, sowie das zu verwendende Protokoll. Um einzelne Ressourcen miteinander verbinden zu können werden Links eingesetzt, welche wiederum eine eindeutige URI für den Zugriff besitzen. Um die Art der vom Service durchzuführenden Aktionen zu definieren gibt es mehrere Standardmethoden, die ebenfalls den Standardmethoden des HTTP Standards entsprechen. Die gängigsten Methoden sind hierbei GET und POST, die weniger gängigen dahingegen sind PUT, DELETE, HEAD und OPTIONS. Gemäß Spezifikation sollte dabei die Methode GET nur Daten abrufen, nicht aber verändern. Um Daten hinzuzufügen, zu ändern oder zu löschen, sollten laut Spezifikation die Methoden PUT und DELETE verwendet werden. Die Methode POST stellt eigentlich eine Ausnahme dar, welche eine eigene Definition der durchzuführenden Aktion erlaubt und damit keine Garantie gibt ob Daten verändert werden, oder nicht. [Til11, Pos. 474ff]

Eine weitere Möglichkeit, Einfluss auf die abgerufenen Daten zu nehmen, ist die Verwendung von Repräsentationen. Dabei wird im Header des HTTP Request der gewünschte Antworttyp mitgeliefert [Til11, Pos. 585]. So kann beispielsweise die Repräsentation eines Mitarbeiterstammsatzes im Format „XML“ einige wichtige Kontaktdaten, hingegen im Repräsentationstyp „Grafik“ das Foto des Mitarbeiters sein.

Genauso wie SOAP oder das darunterliegende HTTP sind die REST Übertragungen zustandslos und damit in jeder Übertragung für sich abgeschlossen und unzusammenhängend mit allen vorrangegangenen und nachfolgenden REST Nachrichten. Um die Sitzungsdaten und ihre Verwaltung muss sich somit ebenfalls die zugrundeliegende Anwendung kümmern. [Til11, Pos. 603]

Eine REST Nachricht besteht daher immer aus einem entsprechenden Header, welcher die anzuwendende Methode den URI und optional die gewünschte Repräsentation beinhaltet. Allfällige Übermittlungsdaten werden im Body der Nachricht übermittelt. Der Server antwortet daraufhin mit einem Statuscode und, gegebenenfalls, je nach angeforderter Methode, mit einem eigenen Body, welcher die Rückgabedaten beinhaltet. Die Statuscodes entsprechen dabei ebenfalls denen des HTTP Standards. Die Codes sind in mehrere Kategorien eingeteilt, welche durch die erste Stelle des dreistelligen Codes kodiert sind [Til11, Pos. 4481ff]:

- 1xx: Informationszwecke (z.B.: 100 – Überprüfung auf Akzeptierung des Inhalts)

- 2xx: Erfolgreiche Verarbeitung (z.B.: 200 – OK, 201 – Created)
- 3xx: Umleitung (z.B.: 301 – Moved Permanently, 307 – Temporary Redirect)
- 4xx: Clientfehler (z.B.: 400 – Bad Request, 404 – Not Found)
- 5xx: Serverfehler (z.B.: 500 – Internal Server Error, 503 – Service Unavailable)

Die Repräsentationsformate definieren den Übertragungstyp des Nachrichten-Body. Die wichtigsten davon sind [Til11, Pos. 1879ff]:

- XML,
- Text (HTML, Plaintext, URI-Listen),
- CSV (Comma Seperated Value),
- JSON (JavaScript Object Notation),
- RSS (Rich Site Summary) und Atom,
- Binäre Formate,
- Microformats und
- RDF (Resource Description Framework).

In vielen Fällen wird das Format HTML unterstützt und als Default-Typ zurückgegeben, wenn keine andere Repräsentation angefordert wurde. Für die Anwendung im Bereich der Cloud Storage Services zur Dateiübertragung sind die binären Formate besonders relevant.

Webbased Distributed Authoring and Versioning (WebDAV)

WebDAV ist eine spezielle Erweiterung des HTTP Befehlssatz um weitere Methoden und Header zur Bearbeitung von Dateien und Darstellung der Datei-Metadaten mittels XML. Es wird inzwischen bereits von einigen Enduser-Anwendungen und Betriebssystemtools nativ unterstützt und kann somit, ohne zusätzliche Softwarekomponenten von vielen Applikationen direkt angesprochen werden. [Til11, Pos. 1440]

Die zusätzlichen HTTP Methoden des WebDAV Protokolls werden in Tabelle 4 kurz beschrieben.

Tabelle 4 - WebDAV Methoden [Til11, Pos. 1439]

WebDAV-Methode	Bedeutung
PROPFIND	Liefert die Eigenschaften (Properties) und deren Werte für eine Ressource zurück
PROPPATCH	Ändert oder entfernt Eigenschaften
MKOL	Legt eine neue Collection-Ressource (analog zu einem Ordner/Verzeichnis) an
COPY	Kopiert eine Collection, Ressource oder Eigenschaft(en)

MOVE	Verschiebt eine Collection oder Eigenschaft(en)
LOCK	Sperrt eine Ressource
UNLOCK	Entsperrt eine Ressource
VERSION-CONTROL	Erzeugt eine Ressource unter Versionskontrolle
REPORT	Liefert Informationen über die Metadaten einer Ressource
CHECKOUT	Muss aufgerufen werden, bevor eine unter Versionskontrolle stehende Ressource modifiziert werden kann
CHECKIN	Erzeugt eine neue Version einer Ressource
UNCHECKOUT	Macht eine CHECKOUT rückgängig
UPDATE	Setzt eine Ressource auf eine andere Version
LABEL	Gibt einer Version einen Namen
MERGE	Konsolidiert zwei Versionen einer Ressource

Eine Abfrage von Ressourcen mittels GET Methode ist im WebDAV Protokoll nicht vorgesehen, dadurch wird der Zugriff über reine REST und HTTP konforme Abfragen erschwert.

3.2 Anbieter und Software

Der Markt rund um die Anbieter von Cloud Storage Services ist inzwischen sehr umfangreich geworden. Die bekanntesten sind Amazon Simple Storage Service [Ama141], Google Drive [Goo144], Microsoft OneDrive (vormals SkyDrive) [Mic143] und Dropbox [Dro14].

Die Nutzungshäufigkeit von Cloud Services auf Ebene der Endanwender wurde von [Wag14] mittels einer Umfrage erhoben und ausgewertet. Die Umfrage ergab, dass der am häufigsten genutzte Service Dropbox (48,6%), der zweithäufigste Google Drive (22,3%) und der dritthäufigste Microsoft OneDrive (19,6%) ist [Wag14, S. 53].

Die Analyse und Lösungserarbeitung dieser Arbeit wird sich daher auf diese drei häufigsten Services beschränken. Die Implementierung des Prototyps erfolgt anschließend für die beiden erstgereihten Services, um eine angemessene Demonstration der Vernetzungsmöglichkeiten zu gewährleisten.

3.2.1 Dropbox

Dropbox wurde 2007 von Drew Houston und Arash Ferdowski gegründet. Die zu Grunde liegende Idee war simpel: Zugriff auf die eigenen Dateien zu jeder Zeit an jedem Ort

[Ame14]. Dropbox wird derzeit von 275 Millionen Menschen und 4 Millionen Unternehmen weltweit eingesetzt und speichert ca. 100 Milliarden Dateien. Jeder Nutzer bekommt einen Gratisspeicherplatz von zwei Gigabyte und kann diesen durch unterschiedliche Promotionsmaßnahmen, wie das Anwerben weiterer User, auf ein Vielfaches erweitern. Es ist ebenfalls möglich, kostenpflichtig zusätzlichen Speicherplatz zu erwerben. [Dro14]

Um Dropbox im lokalen Dateisystem eines PCs zu nutzen, muss am jeweiligen Client eine spezielle Dropbox-Client Software installiert werden. Diese legt dann einen eigenen lokalen Ordner an, welcher alle Dateien des Cloud Speichers beinhaltet. Dabei werden die Dateien und Ordner bei Änderung, Neuanlage oder Löschung automatisch vom Client in den Online Speicherplatz synchronisiert und umgekehrt. Der Datenabgleich kann parallel auf mehreren Dropbox-Clients erfolgen, die jedoch nicht zeitgleich online sein müssen. Die Identifikation der Dateiänderungen und Versionen werden mittels Hashwerten durchgeführt, die zusätzlich in einer lokalen Datenbank gespeichert sind.

Für diese Identifikation wird jede Datei in 4 MB große Abschnitte zerlegt. Die Berechnung der Hashwerte geschieht mittels des SHA-256-Hashalgorythmus. Der Client überträgt geänderte Datenabschnitte nur dann tatsächlich zum Dropbox Service, wenn dieser noch keinen entsprechenden Datenabschnitt mit demselben Hashwert, unabhängig von welchem Anwender, gespeichert hat. Befindet sich bereits ein entsprechender Datenabschnitt in den Speichern des Services, wird lediglich die Information über den Hashwert übertragen und für einen späteren Download der bereits gespeicherte Dateiabschnitt verwendet. Dadurch wird sowohl der Speicherplatz, als auch der anfallende Datenverkehr optimiert, es ergeben sich aber auch mögliche Angriffsvektoren für das unerlaubte Abrufen von Daten (siehe Kapitel 2.4). [Mul11, S. 3]

Zusätzlich zum Zugriff und Abgleich mittels lokal installiertem Dropbox-Client, gibt es die Möglichkeit des Zugriffs über die Weboberfläche von dropbox.com [Dro14] und die Einbettung des Dropbox-Zugriffes in eigene Applikationen mittels Dropbox API [Dro141].

Die Weboberfläche für die Konfiguration und den Dateizugriff der Dropbox ist in Abbildung 8 dargestellt.

Gemäß EU Datenschutz ist die Speicherung personenbezogener Daten in einem Dropbox Speicher grundsätzlich zulässig, da Dropbox dem Safe-Harbor-Programm beigetreten ist (siehe Kapitel 2.4). [Dro142]

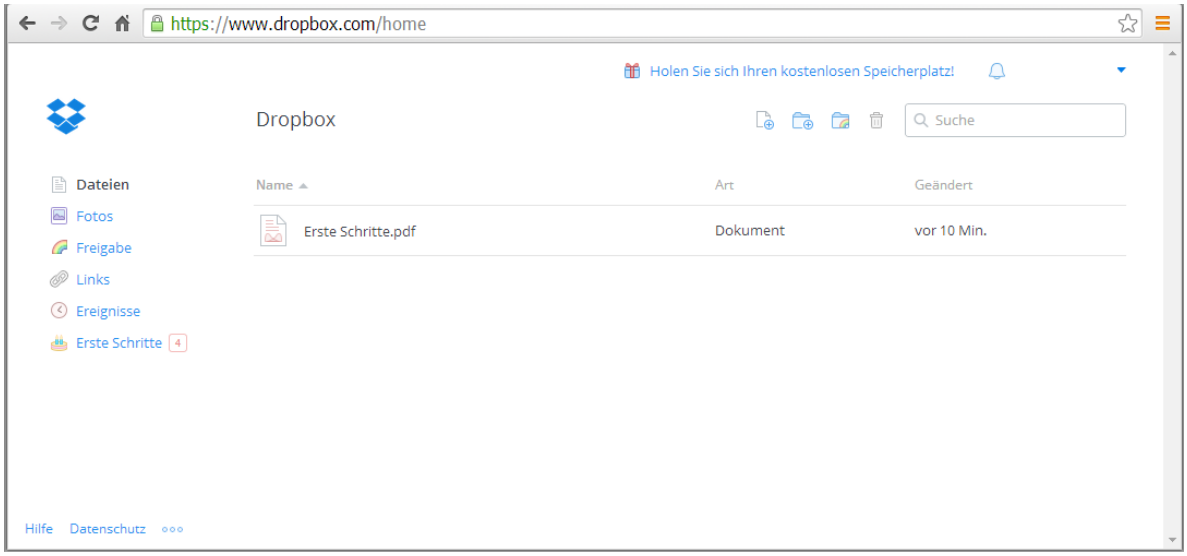

Abbildung 8 - Dropbox Weboberfläche

3.2.2 Google Drive

Google Drive wurde erstmals im April 2012 offiziell angeboten. Vorher gab es Google basierte Speicherdienste nur im Plattformbereich. Jeder Anwender bekommt einen anfänglichen Speicherplatz von 15 Gigabyte zugesprochen, der sich gegen Bezahlung beliebig erweitern lässt. [Hei12] [Goo14]

Im Gegensatz zu Dropbox, werden bei Google Drive geänderte Dateien bei Änderungen immer zur Gänze übertragen und nicht, vor der Betrachtung, in kleinere Blöcke zerlegt. Dadurch ist der Datenverkehr auch entsprechend höher. [Wag14, S. 54]

Der Zugriff auf die Dateien ist sowohl über die Installation einer Clientsoftware, als auch über das Webportal [Goo14] möglich. Bei Nutzung der Clientsoftware integriert sich Google Drive in Form eines eigenen Ordners ins Dateisystem, welcher laufend mit dem Onlinespeicher abgeglichen wird. Bei der Installation des Clients auf mehreren PCs, werden die Daten somit überall synchron gehalten.

Die Oberfläche des Google Drive Webzugriffs ist in Abbildung 9 dargestellt. Soll ein darin befindliches Google Dokument direkt geöffnet werden, steht es automatisch in Google Docs zur Bearbeitung zur Verfügung.

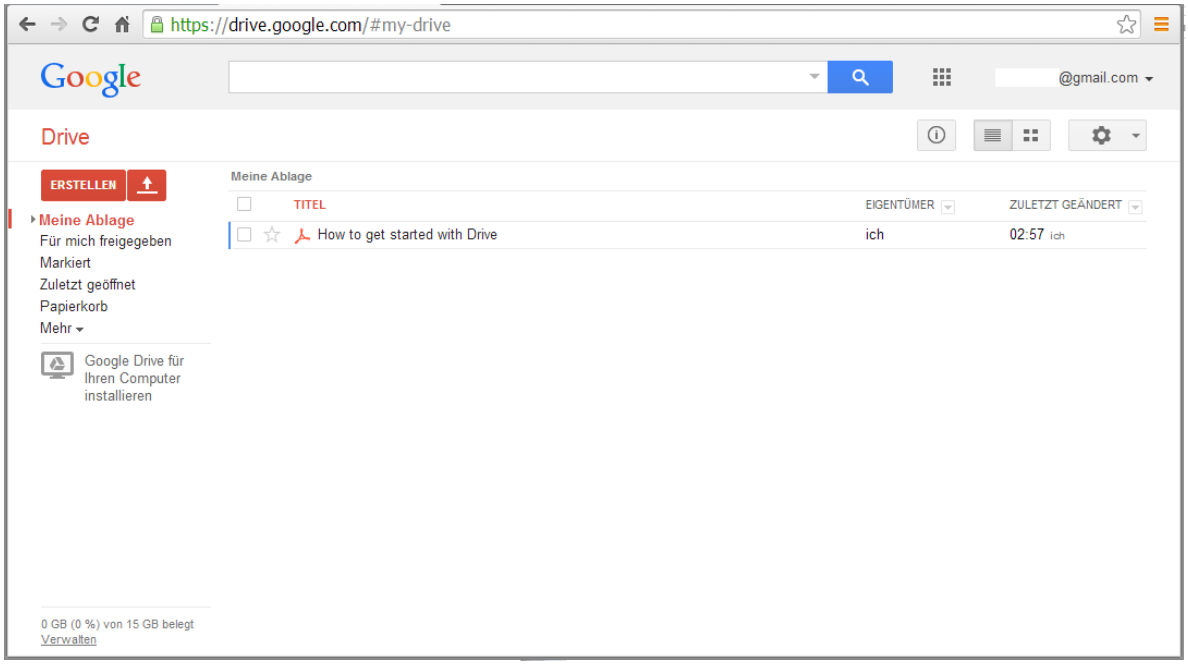

Abbildung 9 - Google Drive Webzugriff

Gemäß EU Datenschutz ist die Speicherung personenbezogener Daten in einem Google Drive Speicher grundsätzlich zulässig, da Google dem Safe-Harbor-Programm beigetreten ist (siehe Kapitel 2.4). [Goo11]

3.2.3 Microsoft OneDrive (SkyDrive)

Microsoft bietet seit April 2012, fast zeitgleich mit Google Drive, seinen Cloud Speicherdienst SkyDrive ebenfalls mit einer Möglichkeit der lokalen Synchronisierung an. Jeder Anwender bekommt initial 7 Gigabyte Speicherplatz gratis zur Verfügung gestellt. Wird zusätzlicher Speicherplatz benötigt, kann dieser auch kostenpflichtig erweitert werden. [Spi12] [Mic143]

Der Online Speicherdienst SkyDrive wurde im Februar 2014 in OneDrive umbenannt. Der Funktionsumfang wurde dabei um einige zusätzliche Funktionen, wie Video-Sharing und eine Kamera-Backup Funktion für Android Geräte, erweitert. [Net141]

OneDrive kann über zwei unterschiedliche Client-Anwendungen auf PCs genutzt werden. Die erste Möglichkeit ist eine Windows 8 App, welche sich nahtlos in das Betriebssystem integriert und jederzeit direkten Zugriff auf den Speicher ermöglicht. Es werden dabei jedoch keine Daten direkt mit dem PC synchronisiert, da der Zugriff immer online erfolgt. Die zweite Möglichkeit ist die Installation des OneDrive Clients. Dieser gleicht die Daten des OneDrive Onlinespeichers mit einer lokalen Ordnerstruktur ab und ermöglicht damit

die Synchronisation der Daten mehrerer PCs. Weiter existiert auch bei OneDrive die Möglichkeit eines Zugriffs über ein Onlineportal [Mic143]. Für den Zugriff von externen Anwendungen und anderen Betriebssystemen als Microsoft Windows ist das Microsoft OneDrive auch über eine WebDAV Schnittstelle ansprechbar. Dies ermöglicht ein leichtes Einbinden in viele Softwareprodukte ohne eine eigene Implementierung über eine spezielle API durchführen zu müssen.

In Abbildung 10 ist die Weboberfläche von OneDrive dargestellt. Sobald eine Datei zum Öffnen ausgewählt wurde, die direkt in einer der Office Apps dargestellt oder bearbeitet werden kann, wird die zugehörige Anwendung gestartet und die Datei darin geöffnet.

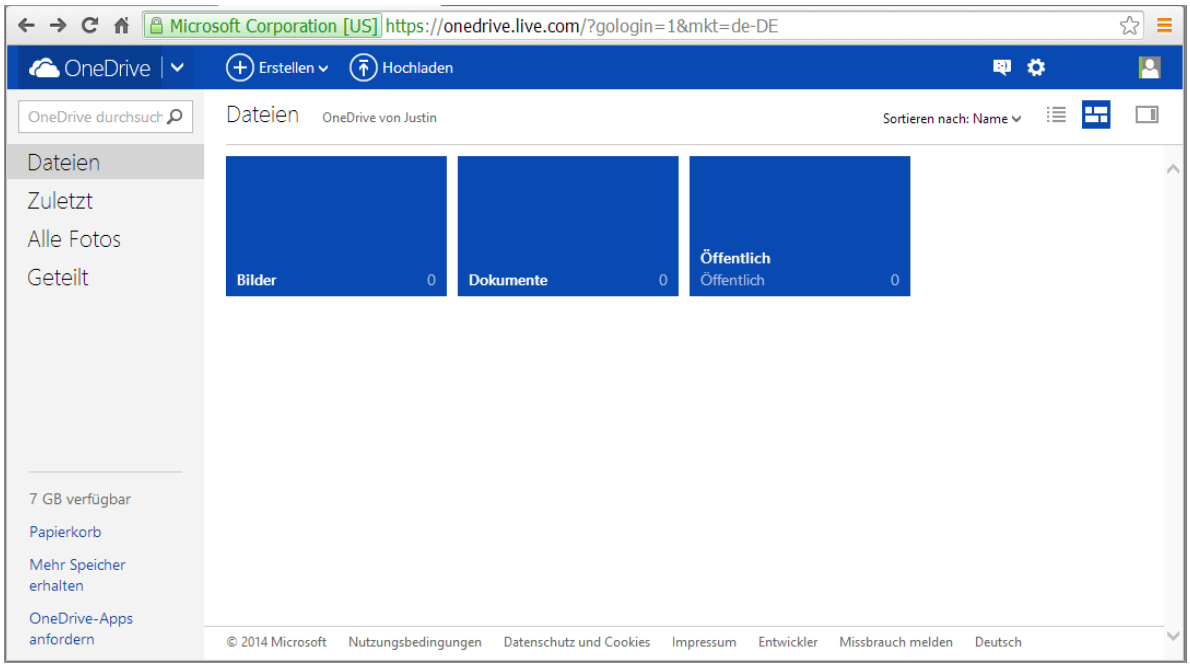

Abbildung 10 - OneDrive Weboberfläche

Genau wie bei Google Drive, werden bei Microsoft OneDrive alle geänderten Dateien komplett übertragen. Ein Zerlegen in kleinere Teile geschieht nicht. Dies führt zu einem höheren Datenverkehr als bei der Synchronisation durch den Dropbox Client. [Wag14, S. 54f]

Gemäß EU Datenschutz ist die Speicherung personenbezogener Daten in einem Microsoft OneDrive Speicher grundsätzlich zulässig, da Microsoft dem Safe-Harbor-Programm beigetreten ist (siehe Kapitel 2.4). [Mic13]

3.3 Application Programming Interfaces (APIs) der Cloud Storage Anbieter

3.3.1 Dropbox

Der Zugriff auf Dropbox geschieht über die, von Dropbox angebotene, REST Schnittstellen-API [Dro143]. Die API ermöglicht einen vollen Zugriff auf alle Dropbox Funktionalitäten, wie Dateiupload, Download und die Abfrage von Dateilisten. Zur Authentifizierung wird OAuth [Mes14] verwendet, welches nach erfolgreicher Überprüfung der Anwenderdaten einen eindeutigen Session Token inklusive Schlüssel bereitstellt.

Eine Übersicht der wichtigsten Methoden ist in Tabelle 5 aufgelistet, die Details zu den einzelnen API Methoden finden sich in Anhang A. Die Zugriffe auf die API erfolgen ausschließlich über SSL Verbindungen. Als Codierung wird UTF-8 verwendet. Die einheitliche Formatierung von Datums- und Uhrzeitformaten entspricht der Form: *"Sat, 21 Aug 2010 22:31:20 +0000"*.

Alle Aufrufe der Dropbox API beginnen mit dem URL *https://api-content.dropbox.com/1/* und werden direkt über HTTP Methoden durchgeführt.

Tabelle 5 - Zusammenfassung der Dropbox API Methoden für Dateioperationen

Methodenaufruf	Beschreibung
GET /files/<root>/<path>	Dateilisten der einzelnen Ordner und Abrufen einer Datei
POST /files/<root>/<path>	Zugriff auf einzelne Dateien
GET /metadata/<root>/<path>	Abruf der Metadaten eines Ordners oder einer Datei
GET /delta	Abfrage des geänderten Daten verglichen mit dem letzten bekannten Stand der Anwendung
POST /fileops/copy	Kopiert einen Ordner oder eine Datei innerhalb der Dropbox
POST /fileops/create_folder	Erstellt einen neuen Ordner
POST /fileops/delete	Löscht einen Ordner oder eine Datei
POST /fileops/move	Verschiebt einen Ordner oder eine Datei
GET /account/info	Liefert Informationen zum Dropbox Account zurück

Bei der Abfrage von Metadaten werden Responsefiles im JSON Format zurückgeliefert. Der Aufbau der Rückgabedatei des Aufrufes */account/info* ist nachfolgend in Tabelle 6 beschrieben.

Tabelle 6 - Dropbox API - JSON Responsefile */account/info*

Element	Beschreibung
referral_link	Aufruf des Services
display_name	Benutzername
uid	Eindeutige ID des Dropbox Accounts
country	Zweistelliger Ländercode des Benutzers
quota_info/normal	Maximal verfügbarer Speicherplatz des Benutzers (in Byte)
quota_info/shared	Belegter Speicherplatz durch freigegebene Ordner (in Byte)
quota_info/quota	Gesamt belegter Speicherplatz des Benutzers (in Byte)

Das Error Handling wird zur Gänze über HTTP Error-Codes abgewickelt (siehe Kapitel 3.1.2). Eine Liste aller, von der Dropbox API eingesetzten, Error Codes befindet sich in Anhang B.

Um eine Verbindung mit der Dropbox API herzustellen, muss zu allererst eine Authentifizierung via OAuth erfolgen. Dazu werden die in Tabelle 7 beschriebenen Methodenaufrufe verwendet.

Tabelle 7 - Zusammenfassung der Dropbox API Methoden für OAuth

Methodenaufruf	Beschreibung
GET /oauth2/authorize	Startet einen neuen Autorisierungsworkflow für einen neuen Dropbox Account
POST /oauth2/token	Liefert einen Access Token für einen bereits verbundenen Dropbox Account zurück
POST /disable_access_token	Deaktiviert einen gültigen Access Token

Für die Einbindung in eigene Applikationen werden Software Development Kits (SDKs) angeboten, welches bereits fertige Implementierungen der API Methoden bereitstellen. Derzeit existieren SDKs für die Programmiersprachen und Entwicklungsumgebunden für Python, Ruby, PHP, Java, Android, iOS und OS X. Für die Einbindungen alle andern Umgebungen kann der Zugriff auch direkt über REST erfolgen. [Dro144]

Um das SDK beispielsweise in PHP Umgebungen zu nutzen muss dieses lediglich am Webserver verfügbar gemacht, und in die Applikation über *require_once "dropbox-sdk/Dropbox/autoload.php";* integriert werden. Danach stehen alle API Methoden direkt in PHP zur Verfügung.

3.3.2 Google Drive

Ähnlich wie Dropbox bietet Google ebenfalls eine API im REST Format an. Die API bietet den vollen Zugriff auf alle Dateioperationen des Google Drives und ermöglicht somit eine volle Integration in eigene Applikationen. [Goo145]

Alle Aufrufe der API Methoden werden relativ zur URI *https://www.googleapis.com/drive/v2* durchgeführt. Eine Übersicht der wichtigsten Methoden ist in Tabelle 8 aufgelistet, die Details zu den einzelnen API Methoden finden sich in Anhang C.

Tabelle 8 - Zusammenfassung der Google Drive API Methoden für Dateioperationen

Methodenaufruf	Beschreibung
POST /files/<fileId>/copy	Kopiert einen Ordner oder eine Datei innerhalb des Google Drive
DELETE /files/<fileId>	Löscht einen Ordner oder eine Datei innerhalb des Google Drive
GET /files/<fileId>	Ruft eine Datei ab
POST /files	Einfügen einer neuen Datei
GET /files	Abrufen einer Dateiliste
PUT /files/<fileId>	Update einer bestehenden Datei
GET /changes/<changeId>	Liefert Informationen zu einer bestimmten Änderung
GET /changes	Gibt eine Liste aller Änderungen zurück
GET /about	Liefert Informationen zum Google Account zurück

Um eine Verbindung mit der Google Drive API herzustellen, muss zu allererst eine Authentifizierung via OAuth erfolgen. Dazu werden die in Tabelle 9 beschriebenen Methodenaufrufe verwendet.

Tabelle 9 - Zusammenfassung der Google Drive API Methoden für OAuth

Methodenaufruf	Beschreibung
GET /oauth2/authorize	Startet einen neuen Autorisierungsworkflow für einen neuen Dropbox Account
POST /oauth2/token	Liefert einen Access Token für einen bereits verbundenen Dropbox Account zurück
POST /disable_access_token	Deaktiviert einen gültigen Access Token

Für den Zugriff über eigene Anwendungen werden von Google eigene API Client Bibliotheken angeboten, welche direkt in die Anwendungen integriert werden können.

Das Einbinden in eine PHP Anwendung erfolgt dann beispielsweise direkt über *require_once 'Google/Client.php';* und die anschließende Verwendung der zugehörigen Klasse *Google_Client*. [Goo146]

3.3.3 Microsoft OneDrive (SkyDrive)

Der Zugriff auf Microsoft OneDrive ist über das offene Protokoll WebDAV möglich. Das bietet wiederum den Vorteil, dass zusätzlich zur Implementierung der OneDrive Anbindung alle Cloud Storages mit WebDAV Schnittstelle kompatibel sind. Die WebDAV Schnittstelle von OneDrive ist über den URL *https://d.docs.live.net/16stelligeOneDriveKennung* erreichbar.

Für die Authentifizierung werden die Benutzerdaten mittels HTTP Basic/Digest Authentifizierung [RFC991] an den Server übergeben:
https://USERID:PASSWD@d.docs.live.net/16stelligeOneDriveKennung.

Zusätzlich zu der WebDAV Schnittstelle gibt es noch eine eigene API für den direkten Zugriff über REST [Mic144].

3.4 Ähnliche Lösungsansätze

3.4.1 OwnCloud

OwnCloud ist ein Open Source Webservice mit dem ein eigener Cloud Server betrieben werden kann. Die Entwicklung von ownCloud begann 2010 und wurde seit dem stetig weitergeführt. Derzeit ist es in der Release Version 6.0 verfügbar. [Own14]

Die Hauptfunktionalitäten liegen in der Bereitstellung eines eigenen Online Storage Services in Form einer privaten Cloud. Zu den aktuellen Basisfeatures zählen derzeit auch eine Kontaktverwaltung, ein Kalender und eine Lesezeichenverwaltung.

Die Basisfeatures können mittels Apps beliebig erweitert werden. Eine davon hat den Namen „External storage support“ und ermöglicht die Einbindung von weiteren Cloud Storages in die Ordnerstruktur des ownCloud Speichers. Es handelt sich dabei jedoch um eine direkte Onlineanbindung und keine Synchronisation mit der ownCloud Ordnerstruktur. Der Server gibt die Anfragen an die externen Speicher somit direkt weiter und übermittelt die Daten dann an den eigenen Clientzugriff [Sch12]. Ein direkter Abgleich zwischen

mehreren Cloud Storage Services ist daher ebenfalls nicht möglich. Eine Verschlüsselungsfunktion für die extern abgelegten Daten ist ebenfalls noch nicht inkludiert.

OwnCloud selbst basiert auf PHP und erfordert mindestens PHP 5.3.3, sowie bis zu 28 PHP Module, bei der Verwendung aller möglichen Funktionalitäten. Als Webserver wird ein Apache Webserver unter Linux empfohlen, es ist jedoch auch jeder andere Webserver mit PHP 5.3.3 Support möglich. Als Datenbank kommt MySQL in beliebiger Version zum Einsatz. [Own141]

Die Hardwareanforderungen beschränken sich damit auf die Lauffähigkeit eines Webservers mit PHP 5.3.3 und MySQL, sowie einem entsprechend großen Datenspeicher. Eine Installation ist daher auch auf stromsparenden Mini-Servern und NAS (Network Attached Storage) Devices möglich.

Für den Zugriff auf OwnCloud steht ein Webportal und eine Client Software, sowie eine Vielzahl an Zusatzentwicklungen der Open Source Community, zur Verfügung. Bei der Clientsoftware handelt es sich um einen Synchronisationsclient, welcher, ähnlich wie der Client von Dropbox (siehe Kapitel 3.2.1), einen Ordner im lokalen Dateisystem des PCs mit dem Onlinespeicher abgleicht.

Zur Anbindung externer Software gibt es eine eigene API und eine WebDAV Schnittstelle.

3.4.2 Jolicloud

Jolicloud wurde 2009 von Tariq Krim and Romain Huet gegründet und hat seinen Sitz in Paris, Frankreich. Das Hauptprodukt trägt den Namen Jolicloud und kombiniert eine Vielzahl von Cloudservices, wie einen Online Storage Service, Multimediaplayer, Office Anwendungen, Newsfeeds und die Vernetzung von Social Network Accounts, innerhalb eines Internetportals. [Jol14]

Eines der Services hat den Namen Jolidrive. Es handelt sich dabei um einen reinen Vernetzungsservice für Cloud Storage Services. Ähnlich wie bei OwnCloud werden hier einzelne Services in eine gemeinsame Struktur eingebunden. Die jeweiligen Cloud Speicher sind dabei jedoch nicht direkt vernetzt, sondern lediglich zentral abrufbar. Eine Synchronisation unterschiedlicher Services oder die Verschlüsselung der abgelegten Daten wird nicht angeboten. Die Oberfläche des Portals ist in Abbildung 11 am Beispiel einer Dropbox Einbindung dargestellt.

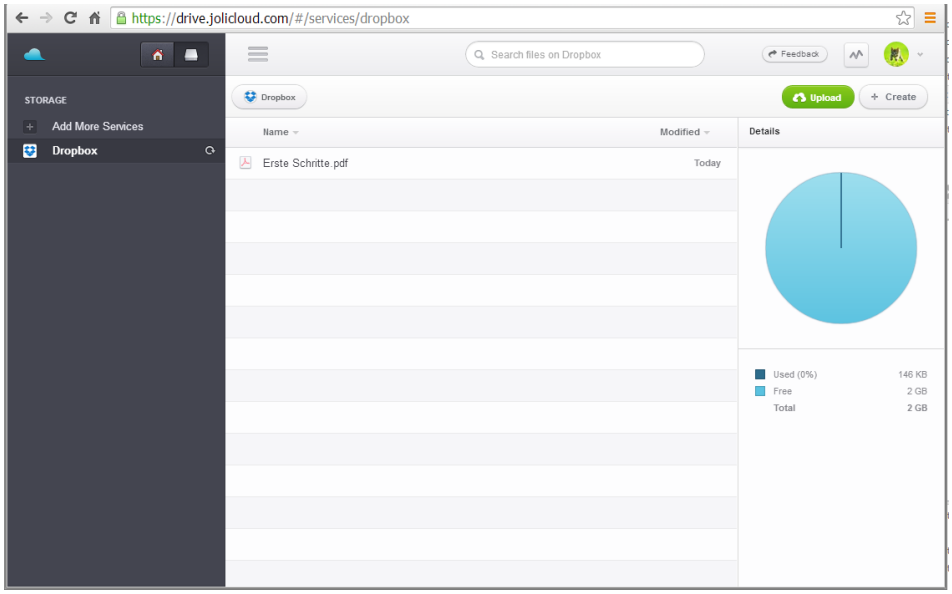

Abbildung 11 - Jolicloud mit Jolidrive

3.4.3 CloudKafe

CloudKafe ist eine ähnliches Portal Service wie Jolicloud. Es ermöglicht die Integration mehrerer Cloud Services in eine gemeinsame Oberfläche. Der Vorteil liegt hier ebenfalls in der gemeinsamen Verwaltung der einzelnen Cloud Speicher und anderer Services, wie Social Networks und Media Services. So ist beispielsweise eine anbieterübergreifende Dateisuche oder die Dateifreigabe von Dateien unterschiedlicher Quellen mittels einheitlicher Abruflinks möglich. Die Daten aus unterschiedlichen Quellen lassen sich dabei in virtuellen Körben, zu sogenannten „Baskets“, zusammenfassen und in dieser Form dann auch mit andern Anwendern teilen. Auch sind Anwendungen zur Betrachtung von Office- und Multimediainhalten in die Oberfläche integriert. [Clo14]

Im Falle der Cloud Storages werden die Services untereinander nicht direkt vernetzt und können weder synchronisiert noch verschlüsselt werden.

Die Oberfläche des Webportals von CloudKafe ist in Abbildung 12 anhand des Beispiels einer Dropbox Integration dargestellt.

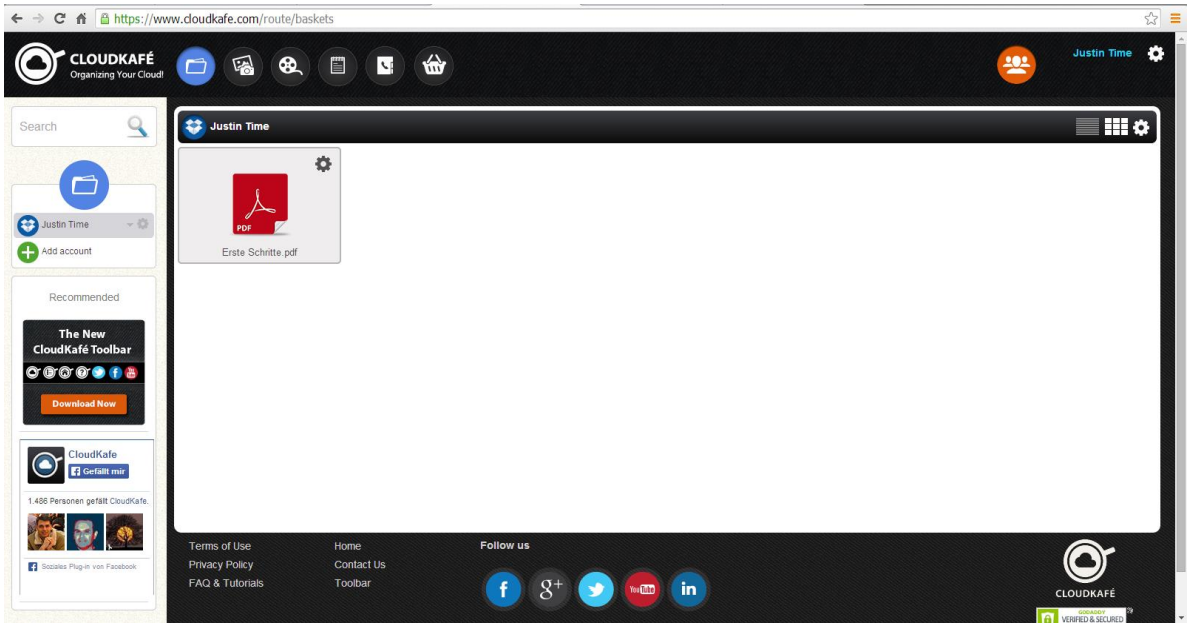

Abbildung 12 - CloudKafe mit Dropbox Integration

3.4.4 ECOCloudS

Bei ECOCloudS (Efficient Combination Of Cloud Storages) handelt es sich um eine Clientapplikation, welche den Clientzugriff und die Ordnersynchronisation mehrerer Cloud Storages kombinieren kann. Die Anwendung befindet sich noch im Stadium eines reinen Prototyps, verfolgt jedoch sehr vielversprechende Ansätze zu Vernetzung und Synchronisation unterschiedlicher Cloud Storage Services, sowie zu deren Datensicherheit durch Verschlüsselung und Speichereffizienz durch Komprimierung. [Hau13, S. 36f]

Die Clientapplikation wurde in C# entwickelt und ist damit unter allen aktuellen Microsoft Windows Umgebungen lauffähig. Eine mobile App ist, aufgrund der Art der Datenstruktur der Anwendung nicht sinnvoll, da eine Installation des Clients auf mehreren Arbeitsplätzen nicht vorgesehen ist. [Hau13, S. 38]

Eine Verteilung der Daten auf mehrere Anbieter und redundante Speicherung ist zentral konfigurierbar. Daher kann diese nur für alle Cloud Storages gemeinsam genutzt werden und lässt sich nicht auf einzelne Ordner beschränken. Bei der Verteilung werden die einzelnen Dateien auf unterschiedliche Anbieter verteilt gespeichert. Die Information, welche Datei auf welchem Cloud Storage verfügbar ist, wird dabei clientseitig verwaltet. Zusätzlich gibt es die Möglichkeit, alle Dateien in allen Speichern synchron zu halten, wobei hier keine Änderungen direkt im Storage des jeweiligen Anbieters erkannt werden, sondern es werden lediglich alle Änderungen durch die Clientapplikation auf alle Anbieter verteilt gespeichert. [Hau13, S. 39ff] Dadurch darf der Anwender jedoch keine

Dateiänderungen direkt bei einem Anbieter durchführen, da sonst die Datenintegrität gefährdet werden würde. Er ist somit auf den Einsatz der Clientsoftware ECOCloudS angewiesen. Auch eine Installation von ECOCloudS auf mehreren Computern ist in diesem Fall problematisch, da die lokalen Datenbanken nicht abgeglichen werden und es hier ebenfalls zu einem Integritätsverlust kommen kann.

Es ist möglich alle Dateien zu komprimieren und zu verschlüsseln. ECOCloudS setzt dazu die Open Source Bibliothek DotNetZip ein. Die Funktionalität selbst und der gewünschte Schlüssel lasst sich dabei nur global für alle verbundenen Cloud Speicher definieren und gilt damit für alle Dateien und Ordner gleichermaßen [Hau13, S. 44]. Es ist somit nicht möglich die Daten an einem vertrauenswürdigen Speicherort unverschlüsselt, und damit für den Zugriff über das jeweilige Onlineportal zugänglich zu machen, und andere Daten auf weniger vertrauenswürdigeren Speicherorten zu verschlüsseln. Gleiches gilt für die Komprimierung, deren Deaktivierung für den direkten Web Zugriff sinnvoll erscheint, auf einem Backupspeicher dahingegen seine Effizienz erhöhen könnte.

In Abbildung 13 ist die Konfigurationsoberfläche von ECOCloudS mit einem verbundenen Dropbox Speicher dargestellt.

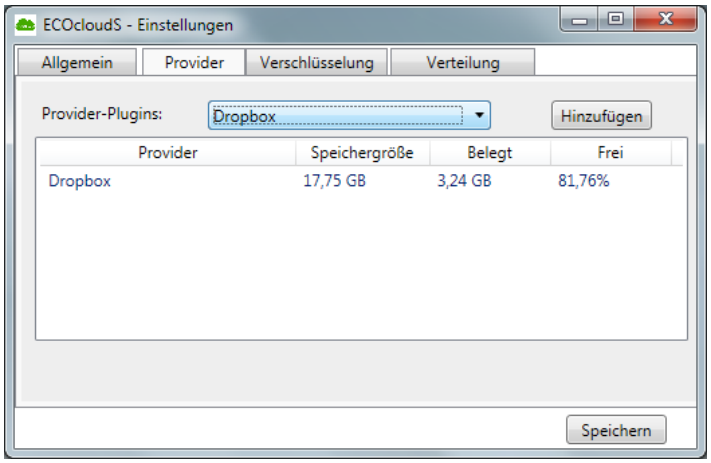

Abbildung 13 - Konfiguration von ECOCloudS mit verbundener Dropbox [Hau13, S. 40]

3.4.5 Vergleich der existierenden Lösungsansätze

Nachfolgend werden nun die vier analysierten Lösungsansätze hinlänglich ihrer Features und dem Anwendernutzen verglichen. Dabei wird ein besonderes Augenmerk auf die Unabhängigkeit und Flexibilität des Anwenders und die Datensicherheit gelegt. Die sich daraus ergebenen Anforderungen wurden in Tabelle 10 aufgelistet und die einzelnen Lösungen damit verglichen.

Tabelle 10 - Vergleich der existierenden Lösungsansätze

Kriterium	OwnCloud	Jolicloud	CloudKafe	ECOCloudS
Private Installation auf eigenem System möglich?	Ja	Nein	Nein	Ja
Weboberfläche vorhanden?	Ja	Ja	Ja	Nein
Clientsoftware vorhanden?	Ja	Nein	Nein	Ja
Schnittstelle zur Integration in eigene Software vorhanden?	Ja	Nein	Nein	Nein
Darstellung für Anwender als integrierter Datenspeicher?	Ja	Nein	Ja	Ja
Synchronisation zwischen den Datenspeichern möglich?	Nein	Nein	Nein	Ja, jedoch mögliche Probleme der Integrität
Verschlüsselung der Datenspeicher möglich?	Nein	Nein	Nein	Ja

Keine der aktuell verfügbaren Lösungen kann alle Anforderungen des Vergleichs erfüllen. Jede hat damit einen speziellen Nutzen für nur einige der möglichen Anwendungsfälle. Der Anwender muss sich dabei zwischen Flexibilität und Datensicherheit entscheiden und im Falle einer gewünschten Synchronisation sogar auf die Anbindung mehrerer Arbeitsplätze verzichten, da ansonsten die Datenintegrität gefährdet würde. Die Entwicklung eines eigenen Lösungsansatzes, welcher alle Anforderungen erfüllt, ist daher am zielbringendsten.

4 Implementierung

4.1 Zielsetzung der Implementierung

Ziel ist es eine eigene Zwischenschicht zur Vernetzung mehrerer Cloud Storages zu entwickeln, welche auf mehrere unterschiedliche Services unterschiedlicher Anbieter zugreift und diese auch untereinander inhaltlich abgleichen kann. Der Name der Anwendung lautet Cloud Storage Pool (CSP).

Die für jeden Anbieter zu implementierenden Zugriffsmethoden sind:

- Ordnerinhalt anzeigen,
- Datei abrufen,
- Neuen Ordner anlegen,
- Neue Datei anlegen,
- Bestehende Datei aktualisieren,
- Ordner löschen und
- Datei löschen.

Die einzelnen Cloud Datenspeicher sollen in Form einer virtuellen Ordnerstruktur abgebildet sein, wobei sich je ein Ordner eines externen Datenspeichers, inklusive seiner Dateien und Unterordner, in mindestens einem virtuellen Ordner des CSP befinden soll. Der Aufbau der virtuellen Ordnerstruktur ist in Abbildung 14 skizziert.

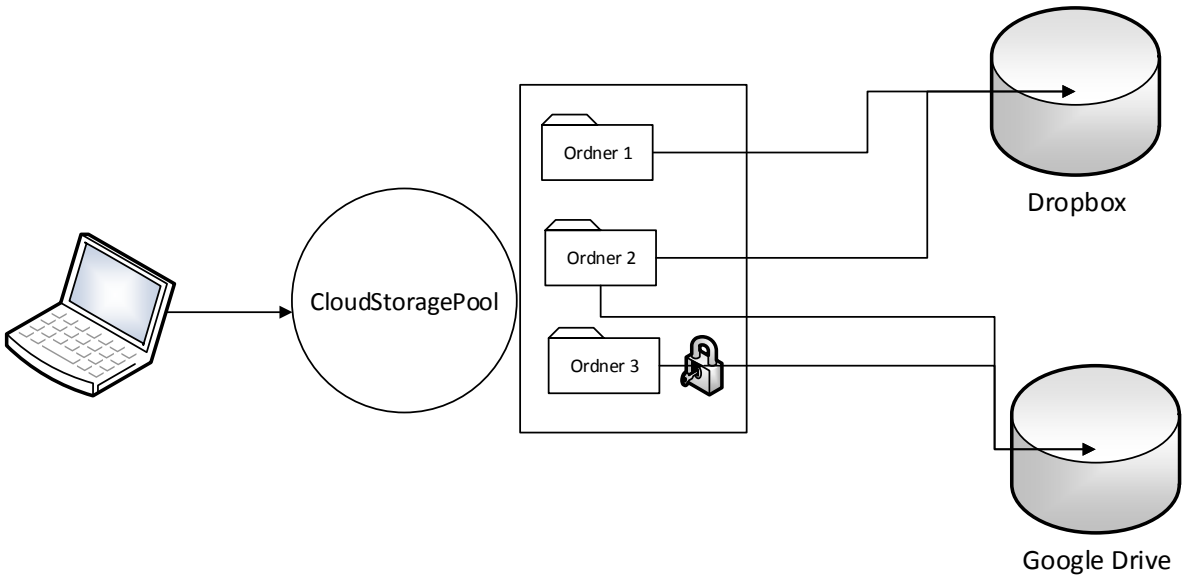

Abbildung 14 - Virtuelle Ordnerstruktur

Für den Zugriff über die Endgeräte des Anwenders wird eine Weboberfläche und eine WebDAV Schnittstelle (siehe Kapitel 3.1.2) implementiert. Damit ist der Zugriff mit nahezu jedem internetfähigen Endgerät und eine Integration in bestehende Dateistrukturen auch betriebssystemunabhängig möglich. Es muss dadurch auch keine spezielle Clientsoftware eingesetzt werden, sondern es kann mittels Browser, oder WebDAV fähiger Applikationen voll auf alle vernetzten Dateien und Ordner zugegriffen werden.

Zusätzlich soll es möglich sein, die einzelnen virtuellen Ordner miteinander abzugleichen, um eine Spiegelung auf beliebig vielen externen Datenspeichern zu ermöglichen. Der Abgleich soll unabhängig von Benutzerinteraktionen möglich sein und im Hintergrund ausgeführt werden. Die Abgleiche mit mehreren externen Datenspeichern werden ebenfalls über virtuelle Ordner abgewickelt, welche untereinander abgeglichen werden. Für den Anwender ist jedoch nur der primäre virtuelle Ordner sichtbar. Die, für den Abgleich vorgesehenen, sekundären virtuellen Ordner werden vor dem Anwender versteckt oder als solche gekennzeichnet.

Zur Gewährung der Vertraulichkeit der in den Cloud Speichern abgelegten Daten, soll eine synchrone Verschlüsselung implementiert werden, um die Daten schnell und einfach verschlüsselt ablegen und entschlüsselt abrufen zu können. Die Verschlüsselung ist dabei für jeden virtuellen Datenspeicher separat aktivierbar und kann auch mit unterschiedlichen Schlüsseln beziehungsweise Passphrasen versehen werden.

Der CSP soll auf einer Großzahl von Servern und Endgeräten einsetzbar und leicht erweiterbar sein. Auch soll eine Integration in bestehende Webservices und Webanwendungen möglich sein. Auch muss natürlich eine Einbindung der Schnittstellen zu den einzelnen Cloudstorage Services möglich sein.

Im Rahmen dieses Prototypen erfolgt eine Anbindung der Cloudstorage Dienste Dropbox und Google Drive, da es sich dabei, in dem Bereich, um die beiden anwenderseitig am weitest verbreiteten Dienste handelt (siehe Kapitel 3.2).

4.2 Architektur des Services

Um die genannten Ziele zu erfüllen muss eine möglichst weit verbreitete Webprogrammiersprache eingesetzt werden, die dabei auf möglichst vielen Serversystemen eingesetzt werden kann.

Eine Erhebung des Einsatzes von serverseitigen Webprogrammiersprachen (Stand 09.05.2014) ergab laut [QSu14] einen Einsatz auf allen erhobenen Servern von:

- 82,0% PHP,

- 17,6% ASP.NET und
- 2,7% Java.

Dabei wurden Server mehrfach in unterschiedlichen Kategorien gezählt, wenn sie mehrere Webprogrammiersprachen unterstützen.

Daher wird auch die Entwicklung des CSP Prototyps in PHP durchgeführt, da so eine möglichst gute Integrierbarkeit in bestehende Webserver und Anwendungen ermöglicht wird.

Der Einsatz einer Datenbank ist zwar nicht unbedingt erforderlich, erleichtert jedoch die Konfiguration und Ablage von Zwischenwerten des Services enorm. Daher wird in dieser Implementierung eine Datenbank eingesetzt. Aufgrund der einfachen Integrierbarkeit und meiner persönlichen Erfahrung wird hierfür die Datenbank MySQL verwendet.

Da es sich um einen Prototypen handelt, wird auf das Design und die Implementierung einer Benutzerverwaltung und eines grafisch ansprechenden Layouts verzichtet. Für die Authentifizierung des Anwenders wird eine Basisauthentifizierung des Webservers verwendet. Die Authentifizierung und Verschlüsslung der Datenübertragung vom Anwenderclient zum CSP erfolgt über die SSL Verbindung des Webservers.

4.2.1 Programmstruktur

Die Struktur der Anwendung ist in mehreren Anwendungsdateien Ordnern abgebildet. Die eigenen Klassendateien befinden sich im Verzeichnis *classes*, die notwendigen API Pakete im Verzeichnis *api* und die vom Anwender direkt anzusprechenden Programmdateien und ihre zugehörigen Includedateien direkt im Hauptverzeichnis. Zusätzlich gibt es noch einen Ordner mit dem Namen *tmp*, indem temporäre Übertragungsdateien nötigenfalls zwischengespeichert werden.

Alle Programmordner werden auf Seiten des Webservers von direkten Anwenderzugriffen geschützt. Ein Zugriff auf die beinhaltenden Dateien ist nur durch die Anwendungen selbst, nicht jedoch durch externe Serveranfragen möglich. Dies ist wichtig, um die temporären Daten und die API Daten vor unerwünschten Zugriffen zu schützen.

Für die Abbildung der virtuellen Ordnerstruktur und der Implementierung des WebDAV Servers werden eigene Klassen entwickelt, deren Aufbau und Funktionsweise nachfolgend in Kapitel 4.2.2 näher beschrieben sind.

Die Anwendung selbst hat mehrere direkte Programmeinstiegspunkte für den Zugriff des Anwenders:

- index.php Weboberfläche des CSP,
- getfile.php Dateidownload über die Weboberfläche des CSP,
- webdav.php WebDAV Schnittstelle des CSP,
- syncall.php Synchronisation der CSP Datenspeicher,
- dropbox_auth.php Autorisierung des CSP gegenüber Dropbox und
- googledrive_auth.php Autorisierung des CSP gegenüber Google Drive.

Die Basiskonfiguration des CSP und allgemein verwendete Funktionen befinden sich in den Dateien:

- config.inc.php und
- functions.inc.php.

Dabei handelt es sich um die Konfiguration der Anwendungsidentifikation gegenüber den Cloudstorage Services und den Zugriffsdaten für die Datenbank. Weiter wurden hier Methoden zur einfachen Instanziierung von Datastorage Klassen (siehe nachfolgend in Kapitel 4.2.2) abgebildet. Diese sind notwendig, um die Abfrage der Parameter des virtuellen Ordners und des damit verbundenen externen Datenspeichers aus der Datenbank für alle Anwendungsteile zentral abzubilden. Eine Instanziierung der Zugriffsklassen ist dadurch durch die einfach Übergabe des Namens oder der Datenbank Identifikation des virtuellen Verzeichnisses möglich.

Der Sourcecode der Includedateien befindet sich in Anhang D.1. und D.2. dieser Arbeit.

4.2.2 Klassenstruktur

Die Klassenstruktur des CSP besteht aus einer Klasse für den WebDAV Zugriff und den Klassen für die Verbindung der externen Cloud Datenspeicher in die virtuelle Ordnerstruktur. Die Klasse für den WebDAV Zugriff ist in Kapitel 4.7 genauer beschrieben.

Da die externen Datenspeicher in Form einer virtuellen Ordnerstruktur in den CSP eingebunden werden sollen, wird hierfür eine abstrakte Klasse *Datastorage* verwendet, welche die allgemeingültige Struktur für die Zugriffe der CSP Anwendungen vorgibt. Aus ihr werden wiederum die Klassen für den Zugriff auf den jeweiligen Cloud Speicherdienst abgeleitet und die abstrakten Methoden fertig implementiert. Im Rahmen dieses Prototypen gibt es daher folgende Klassen für den Aufbau der virtuellen Ordnerstruktur:

- Datastorage.php abstrakte Klasse der CSP Ordner,
- Datastorage_Dropbox.php Implementierung der Dropbox Zugriffe und
- Datastorage_Googledrive.php Implementierung der Google Drive Zugriffe.

Die zu implementierenden Klassenmethoden sind:

- __construct Konstruktor zur Initialisierung des Datenspeichers,

- getName Abfrage des Namens des virtuellen Verzeichnisses,
- getFilelist Abfrage der Dateien eines Verzeichnisses,
- getFile Abrufen einer Datei aus dem externen Datenspeicher,
- uploadFile Hochladen einer Datei in den externen Datenspeicher,
- uploadFileStream Hochladen eines Streams in den externen Datenspeicher,
- deleteFile Löschen einer Datei oder eines Ordners,
- copyFile Kopieren oder Verschieben einer Datei oder eines Ordners,
- createFolder Anlegen eines neuen Ordners,
- getDelta Abfrage von Dateiänderungen des externen Datenspeichers.

Zusätzlich gibt es zwei Methoden zur Ver- und Entschlüsselung der Dateiübertragungen zu den externen Datenspeichern:

- csp_encryptStream und
- csp_decryptStream.

Diese sind bereits in der abstrakten Klasse Datastorage ausprogrammiert und können gegebenenfalls auch in den abgeleiteten Klassen überlagert werden, da sie nicht als final deklariert sind und eine Überlagerung, zur Änderung des Kryptographie-Verfahrens einzelner externer Datenspeichertypen, durchaus notwendig sein kann. Die beiden Methoden und ihre Funktionsweise sind in Kapitel 4.5 genauer beschrieben.

In der Implementierung des Prototyps sind die Klassen für den Dropbox und Google Drive Zugriff ausprogrammiert. Eine Anbindung an weitere Speicherdienste ist durch weitere Implementierungen von Ableitungen der Klasse Datastorage jederzeit möglich. Welche Art von Speicherdienst sich darin befindet ist für die restliche Anwendung irrelevant. Es wäre daher auch möglich, Zugriffe auf Datenbankspeicher oder lokale Daten zu integrieren. Das Klassendiagramm der Klasse Datastorages ist nachfolgend in Abbildung 15 dargestellt.

Da in Google Drive kein direkter Zugriff auf Dateipfade möglich ist, wurden hier zusätzlich die Klassenmethoden getMetadataByPath und getPathById implementiert, die den benötigten Dateipfad aufgrund der Dateiverknüpfungen in Google Drive ermitteln. Hierbei muss zur Ermittlung eines Dateipfades die Zugehörigkeit eines Objektes zu seinem übergeordneten Ordnerobjekt jeweils einzeln für jede Ebene des Pfades bis zur Wurzel abgefragt werden. Für die Ermittlung eines Objektes anhand des Pfades muss dieses wiederum von der Wurzel des Google Drive Speichers weg, durch das Durchlaufen der beinhaltenden Ordnerobjekte der einzelnen Strukturebenen, gesucht werden. Diese Vorgehensweise ist notwendig, da der Zugriff der virtuellen Datenspeicher über die Verwendung von Orderstrukturen erfolgt, um einen Abgleich und eine virtuelle Einbindung gemeinsam mit anderen Datenspeicherdiensten zu ermöglichen. Die Funktionsweise der Pfadzugriffe über die Klassenmethoden getMetadataByPath und getPathById sind in Abbildung 16 skizziert.

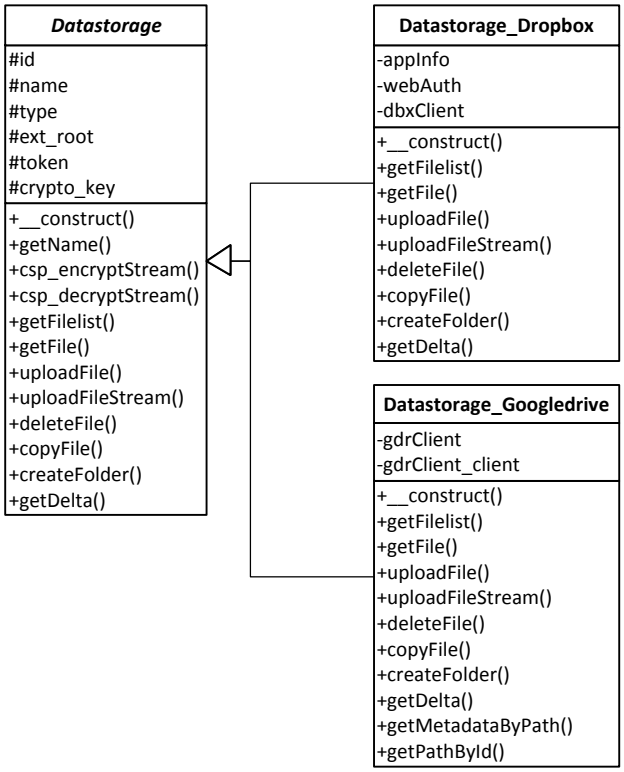

Abbildung 15 - Klassendiagramm der Datastorage-Klassen

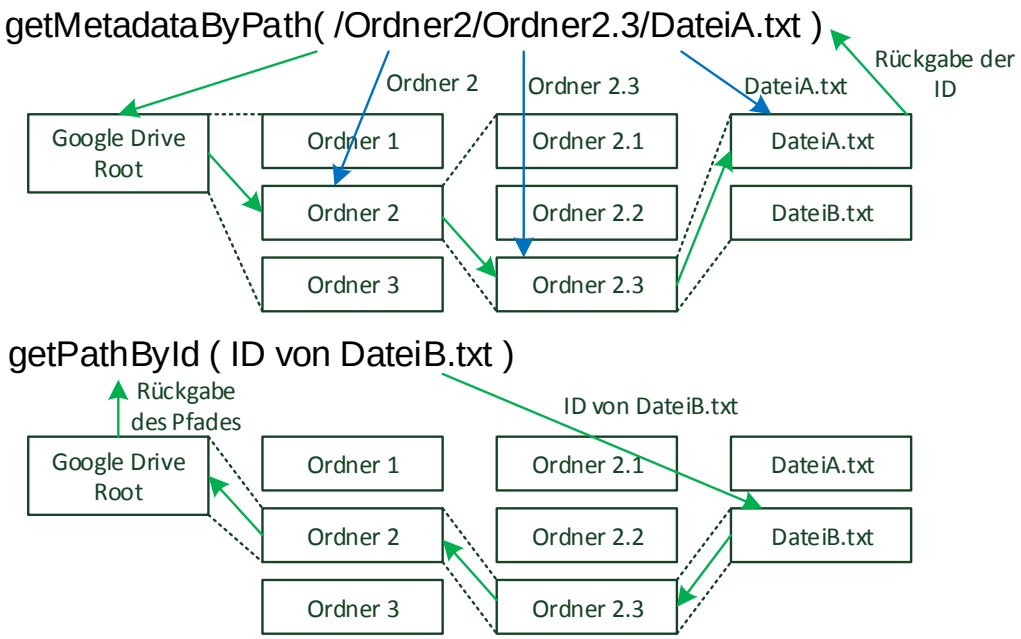

Abbildung 16 - Google Drive Zugriff über Dateipfade

Der Sourcecode der Implementierung der vier Klassen befindet sich in Anhang D.9., D.10., D.11. und D.12.

4.2.3 Datenbankmodell

Die virtuelle Verzeichnisstruktur und die nötigen Parameter für die Verbindung zu den externen datenspeichern und ihrer Synchronisation werden in einer Datenbank abgelegt. Das Datenmodell besteht aus zwei Tabellen:

- csp_datastorages virtuelle Ordnerstruktur und der Verbindungen sowie
- csp_datastorages_types mögliche Typen der externen Datenspeicher.

Derzeit gibt es die möglichen Typen DRPBO für Dropbox Verbindungen und GOODR für Google Drive Verbindungen der Datenspeichern in der Tabelle csp_datastorage_types.

Nachfolgend, in Abbildung 17, ist das Entity Relationship Modell (ER Modell) der Datenbanktabellen grafisch dargestellt.

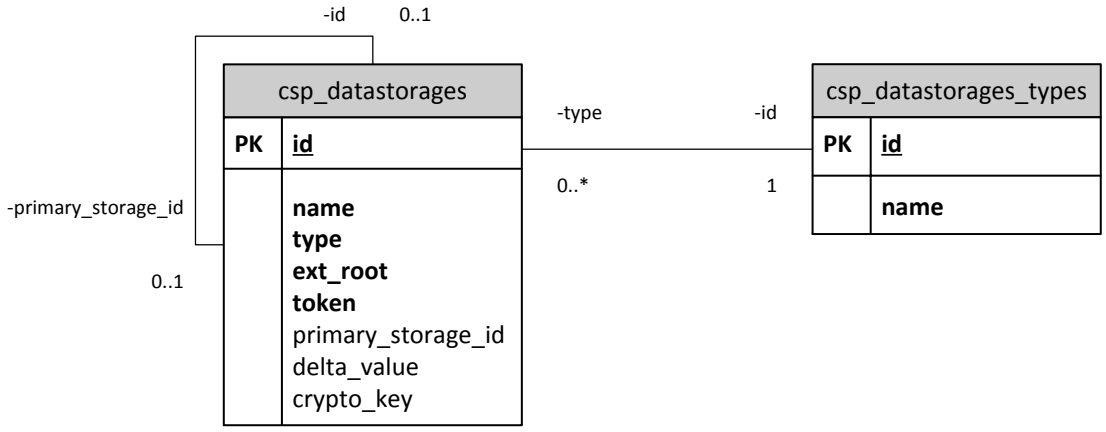

Abbildung 17 - ER Modell der CSP Datenbank

Der Sourcecode der Implementierung der Datenbankverbindungen sowie die SQL Statements zur Generierung der Tabellen befinden sich in Anhang D.1.

4.3 Autorisierung der Cloud Storage Services

Für die Verknüpfung des CSP mit den einzelnen Cloud Datenspeichern ist es erforderlich, eine Autorisierung gemäß der Vorgaben des jeweiligen Anbieters durchzuführen und diese in der CSP Konfiguration zu hinterlegen. Bei Dropbox und Google Drive handelt es sich dabei um eine OAuth2 Autorisierung, die in beiden Fällen ähnlich abläuft.

Zuerst ist es notwendig die Anwendung CSP beim Anbieter zu registrieren, damit sie entsprechend ihrem Verwendungszweck mit dem Anbieter der Anwendung verknüpft ist und ihre Anforderungen akzeptiert werden. Nach der Registrierung wird der Anwendung eine eindeutige Identifikation und ein Schlüssel zugewiesen.

Zur Verknüpfung der Anwendung mit einem bestimmten Datenspeicher Account wird ein Workflow eingeleitet, in dem der Anwender den Wunsch der Verknüpfung bestätigt und über die angeforderten Rechte der Anwendung direkt vom Storageservice informiert wird. Nach der Bestätigung wird ein Token generiert, der im CSP hinterlegt werden muss. Ab dann ist ein direkter Zugriff der Anwendung CSP auf den Datenspeicher gestattet.

4.3.1 Autorisierung gegenüber Dropbox

Um eine Autorisierung durchführen zu können, muss der CSP zuerst als Dropbox Anwendung registriert werden. Dies geschieht in die Dropbox App-Console [Dro145] über den Button „Create app“ (siehe Abbildung 18).

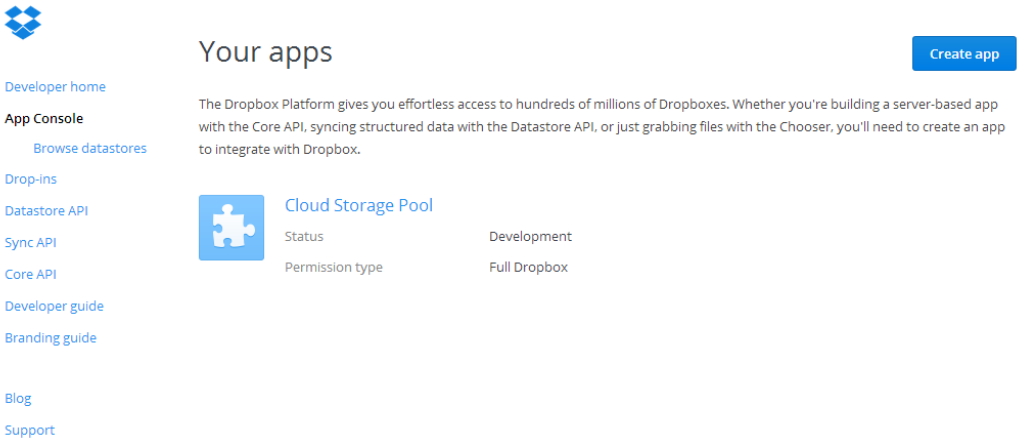

Abbildung 18 - Dropbox App Console

Um eine Anwendung zur Verwendung der API Methoden zu erstellen, muss die Erstellung einer „Dropbox API app“ mit der Option „Files and Datastores“ ausgewählt werden. Für den Zugriff auf alle im Datenspeicher vorhandenen Dateien empfehlen sich zusätzlich bei der Frage „Can your app be limited to its own folder?“ die Optionen „No - My app needs access to files already on Dropbox.“ und bei „What type of files does your app need access to?“ die Option „All file types“ auszuwählen, da sonst ein eigener Speicherbereich für die Anwendung angelegt wird und diese keinen Zugriff auf die Anwenderdateien erhalten würde (siehe Abbildung 19).

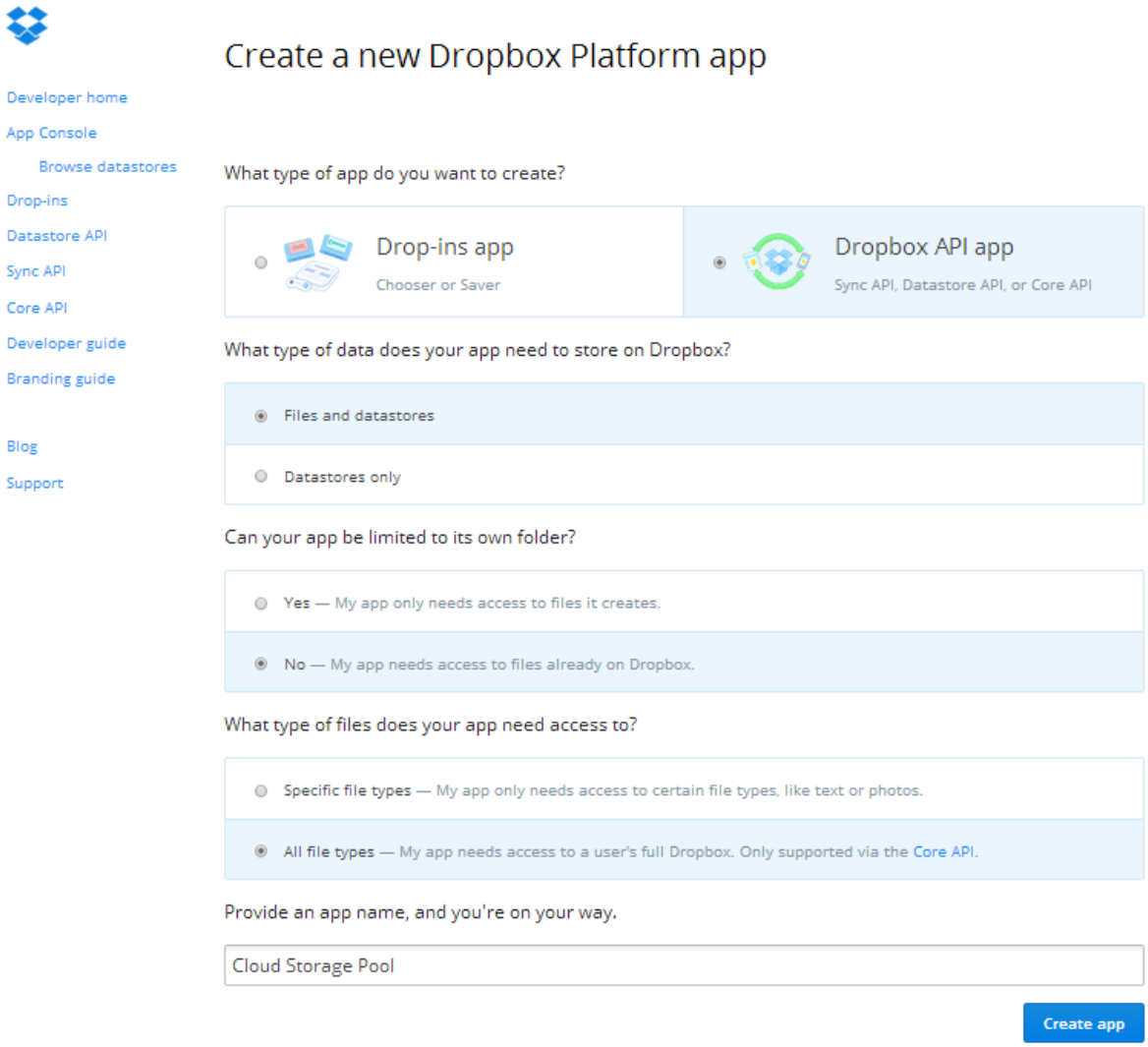

Abbildung 19 - Neue Dropbox Anwendung anlegen

Danach ist die Anwendung angelegt und alle notwendigen Parameter sind auf der Übersichtsseite der App abrufbar und konfigurierbar (siehe Abbildung 20). Die wichtigsten Parameter sind der App-Key und das App-Secret. Diese beiden Werte müssen in der Konfigurationsdatei config.inc.php des CSP eingetragen sein, damit sie für alle Dropbox Zugriffe der Anwendung verfügbar sind.

Die Autorisierung der Anwendung zur Verknüpfung mit einem Dropboxspeicher erfolgt nun direkt über die API. Dazu wurde die Programmdatei dropbox_auth.php entwickelt, welche den Anwender durch die nötigen Schritte führt.

In der API wird zuallererst eine Autorisierungs-URL abgerufen, den der Anwender aufrufen muss. Dort wird er von Dropbox über die Berechtigungen der Anwendungen informiert und kann diese bestätigen. Nach der Bestätigung wird ein Code zurückgegeben, der wiederum in das Formular der CSP Anwendung eingetragen werden muss. Diese fordert dann einen

Token an und gibt ihn im Formular aus. Der Token muss anschließend nur noch in der Tabelle csp_datastorage beim zu verknüpfenden Datenspeicher eingetragen werden. Die Zugriffe des CSP erfolgen danach automatisch direkt unter Verwendung des Tokens.

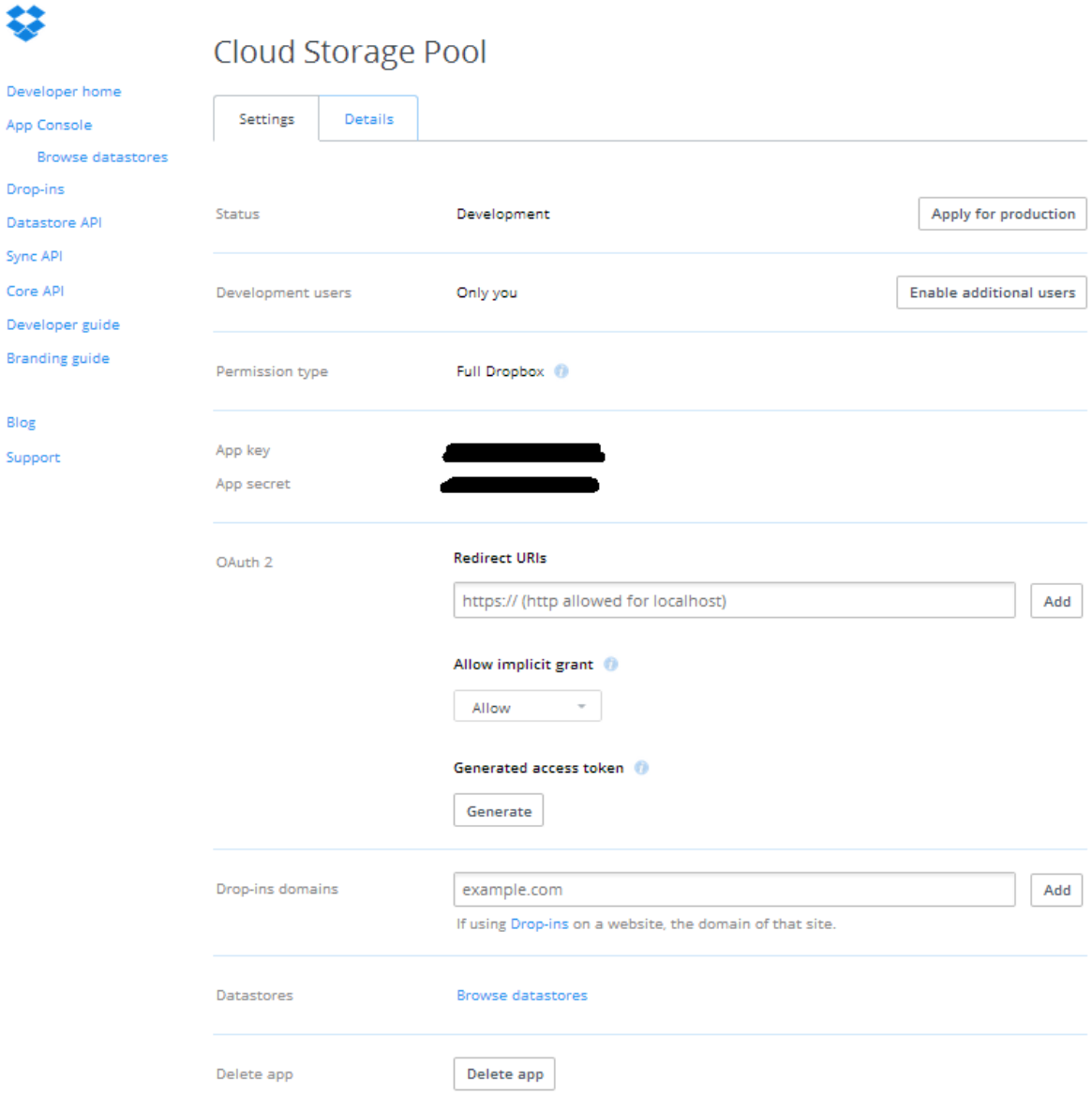

Abbildung 20 - Dropbox Applikation Einstellungen

In Abbildung 21 ist die Benutzeroberfläche der CSP Programmdatei dropbox_auth.php dargestellt.

Abbildung 21 - CSP Dropbox Autorisierung

Der Sourcecode der Implementierung der Dropbox Autorisierung befindet sich in Anhang D.3.

4.3.2 Autorisierung gegenüber Google Drive

Die Autorisierung funktioniert bei Google Drive ähnlich wie bei Dropbox. Zuerst muss die Anwendung als Google Anwendung registriert werden. Dies geschieht bei Google über die Developers Console [Goo147] und über die Auswahl „Create new project“. Sind bereits Projekte vorhanden, kann hier auch eines davon zur Bearbeitung ausgewählt werden (siehe Abbildung 22).

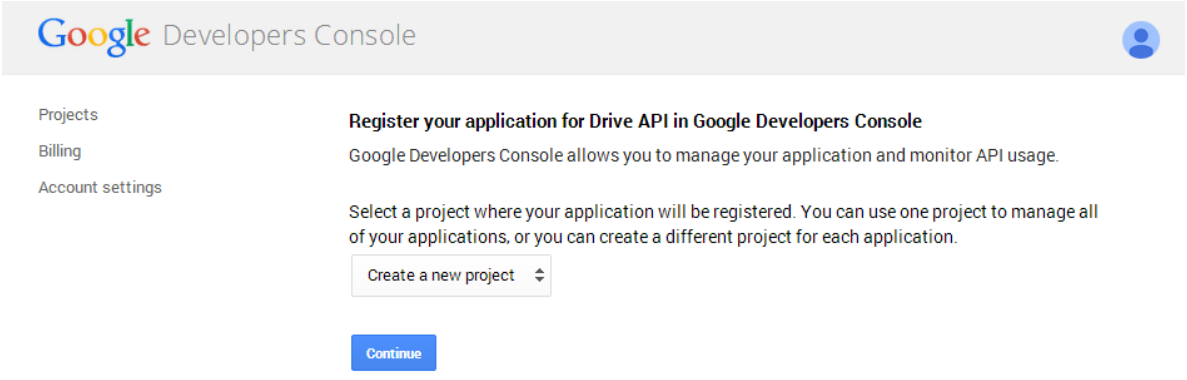

Abbildung 22 - Google Developers Console

Unmittelbar nach der Auswahl von „Continue“ wird nun ein neues Google API Projekt angelegt, welches auch schon eine Client-ID zugewiesen bekommt. Unter dem Menü API muss nun noch der Zugriff für die „Drive API“ und das „Drive SDK“ aktiviert werden (siehe Abbildung 23).

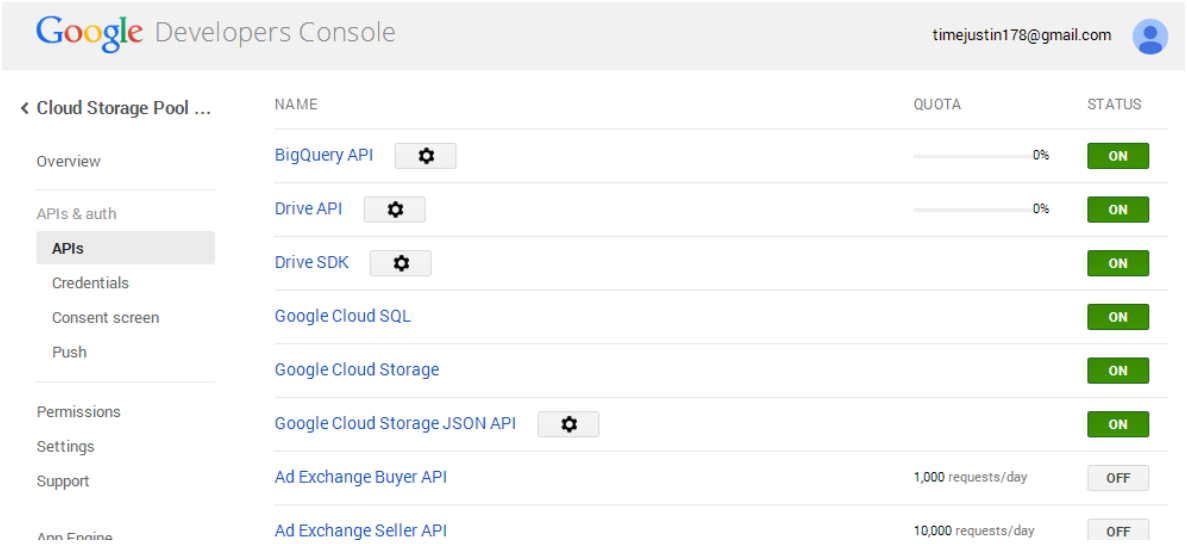

Abbildung 23 - Google Developers Console APIs

Unter dem Menüpunkt Credentials muss nun mittels „Create new client id“ eine neue Client-ID für den Zugriff des CSP angelegt werden. Die Auswahl hierbei sind „Web application“ als Applikationstyp und die Eingabe der URL des CSP unter „Authorized Javascript Origins“. In das Feld „Authorized Redirect URI“ muss der URI zur Programmdatei googledrive_auth.php angegeben werden, da dieser Pfad später zur Übergabe des Tokens verwendet wird. Nach dem Speichern findet sich nun ein neuer Client ID Eintrag in der Liste, welcher die Werte „Client ID“ und „Client secret“ enthält, welche wiederum in der Konfigurationsdatei config.inc.php des CSP eingetragen werden müssen, um eine Verbindung der Anwendung mit Google Drive zu ermöglichen (siehe Abbildung 24).

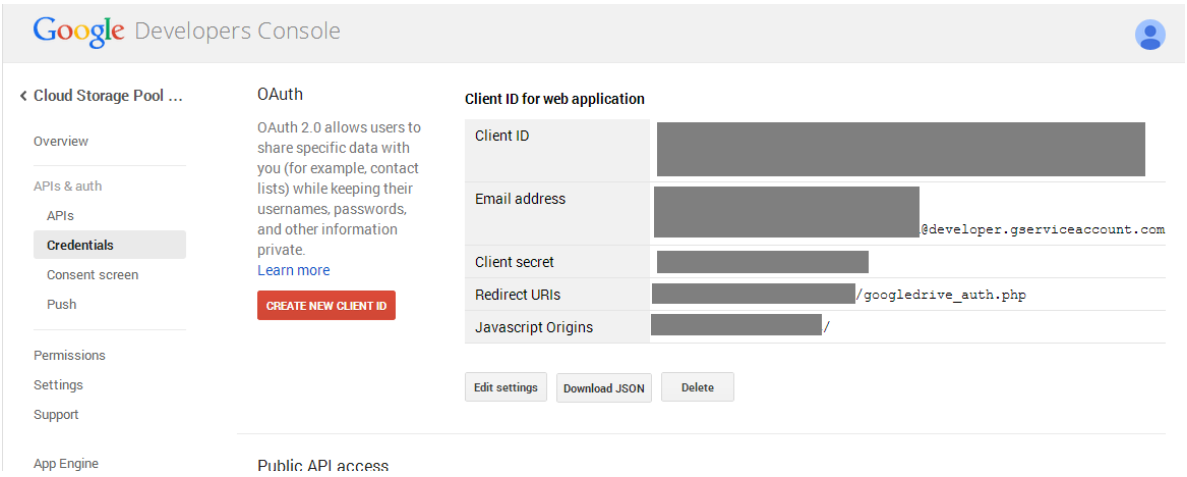

Abbildung 24 - Google Anwendung Client ID

Von der Anwendungsdatei googledrive_auth.php der API wird nun ebenfalls ein Autorisierungs-URL abgerufen, den der Anwender aufrufen muss. Dort wird er von Google Drive über die Berechtigungen der Anwendungen informiert und kann diese bestätigen. Nach der Bestätigung wird ein Redirect auf den URI der Anwendung googledrive_auth.php ausgeführt, welcher den Autorisierungscode übergeben Dieser fordert dann einen Token an und gibt ihn dem Anwender aus. Der Token muss anschließend nur noch in der Tabelle csp_datastorage beim zu verknüpfenden Datenspeicher eingetragen werden. Die Zugriffe des CSP erfolgen danach automatisch direkt unter Verwendung des Tokens.

In Abbildung 25 ist die Benutzeroberfläche der CSP Programmdatei dropbox_auth.php dargestellt.

Abbildung 25 - CSP Google Drive Autorisierung

Der Sourcecode der Implementierung der Google Drive Autorisierung befindet sich in Anhang D.4.

4.4 Synchronisation

Die Synchronisation erfolgt über die Informationen der jeweiligen Cloud Storage Services. Dazu wird im CSP für jeden verbundenen virtuellen Ordner eine Dateiliste mit den zugehörigen Änderungen abgerufen. Die Datenspeicher werden dabei immer als primärer und als sekundärer Datenspeicher betrachtet, um mögliche Konflikte behandeln zu können. Dabei bekommt immer der primäre Datenspeicher Vorrang gegenüber dem sekundären Datenspeicher. Die alte Datei des sekundären Datenspeichers wird dabei als Konfliktdatei mit dem Zusatz „KONFLIKT_TIMESTAMP“ beibehalten und wiederum auf beiden Datenspeichern abgelegt.

Die Änderungslisten werden in regelmäßigen Intervallen vom CSP Server durch den Aufruf der Anwendungsdatei syncall.php abgeglichen. Dabei wird immer von den Datenspeichern ausgegangen, die selbst keinen primären Datenspeicher zugewiesen haben und somit im Abgleich die höchste Priorität einnehmen. Danach wird der Vergleich für den möglichen weiteren sekundären Datenspeicher hinter dem zuerst betrachteten Datenspeicher

fortgesetzt, und so weiter, bis der letzte sekundäre Datenspeicher in der Referenzkette erreicht ist. Dieser Zusammenhang ist in Abbildung 26 dargestellt.

Der Datenspeicher „Datastorage 1“ ist hierbei der primäre Datenspeicher erster Ebene, ihm nachgeordnet ist der Datenspeicher „Datastorage 2“ und diesem wiederum nachgeordnet ist „Datastorage 3“. Der Speicher „Datastorage 4“ hat hingegen keinen sekundären Datenspeicher und wird somit nicht synchronisiert. Für den Anwender gibt es hier zwei primäre virtuelle Ordner: „Datastorage 1“ und „Datastorage 4“. Die Verzeichnisse „Datastorage 2“ und „Datastorage 3“ sind vor dem Anwender versteckt, beziehungsweise als sekundäre Ordner gekennzeichnet.

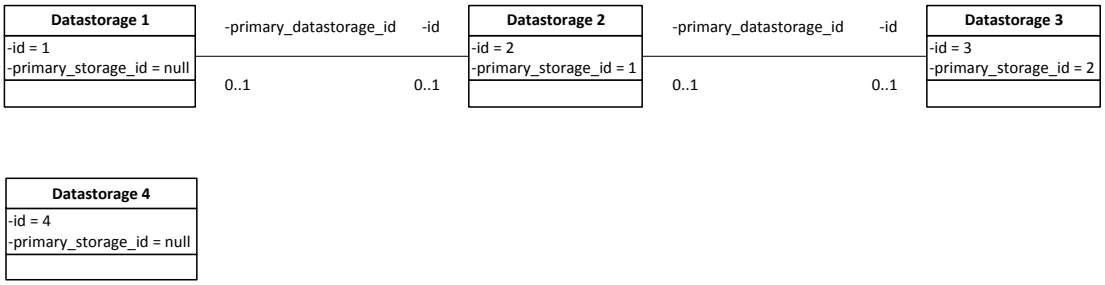

Abbildung 26 - Beispiel der Verbindung der Datastorages für den Datenabgleich

Damit beim Abgleich einer Referenzkette mehrerer Datenspeicher auch geänderte Daten der vorrangegangenen Abgleiche weitergegeben werden, werden alle Abgleiche in einem temporären Zwischenspeicher abgelegt und für jeden sekundären Datenspeicher zusätzlich ausgeführt. Damit wird sichergestellt, dass jede Änderung auf allen Datenspeichern des Synchronisationsdurchgangs ausgeführt wird. Diese Abgleiche werden solange wiederholt, bis keine neuen Dateiänderungen mehr gefunden werden. Dadurch wird verhindert, dass anwenderseitige Dateiänderungen, die während eines Synchronisationsdurchlaufs gemacht wurden, ignoriert werden.

Sobald mit dem nächsten primären Datenspeicher erster Ebene fortgesetzt wird, wird dieser Zwischenspeicher wieder geleert und im Durchlauf der neuen Referenzkette wieder befüllt. Der detaillierte Ablauf der Synchronisationsdurchläufe ist in Abbildung 27 in Form eines Aktivitätsdiagrammes dargestellt.

Die Ausgabe der Programmdatei syncall.php erfolgt, anstatt in HTML Code, als Textdatei, um eine bessere Integration in Logdateien zu ermöglichen. Der Sourcecode der Implementierung der Synchronisation befindet sich in Anhang D.7.

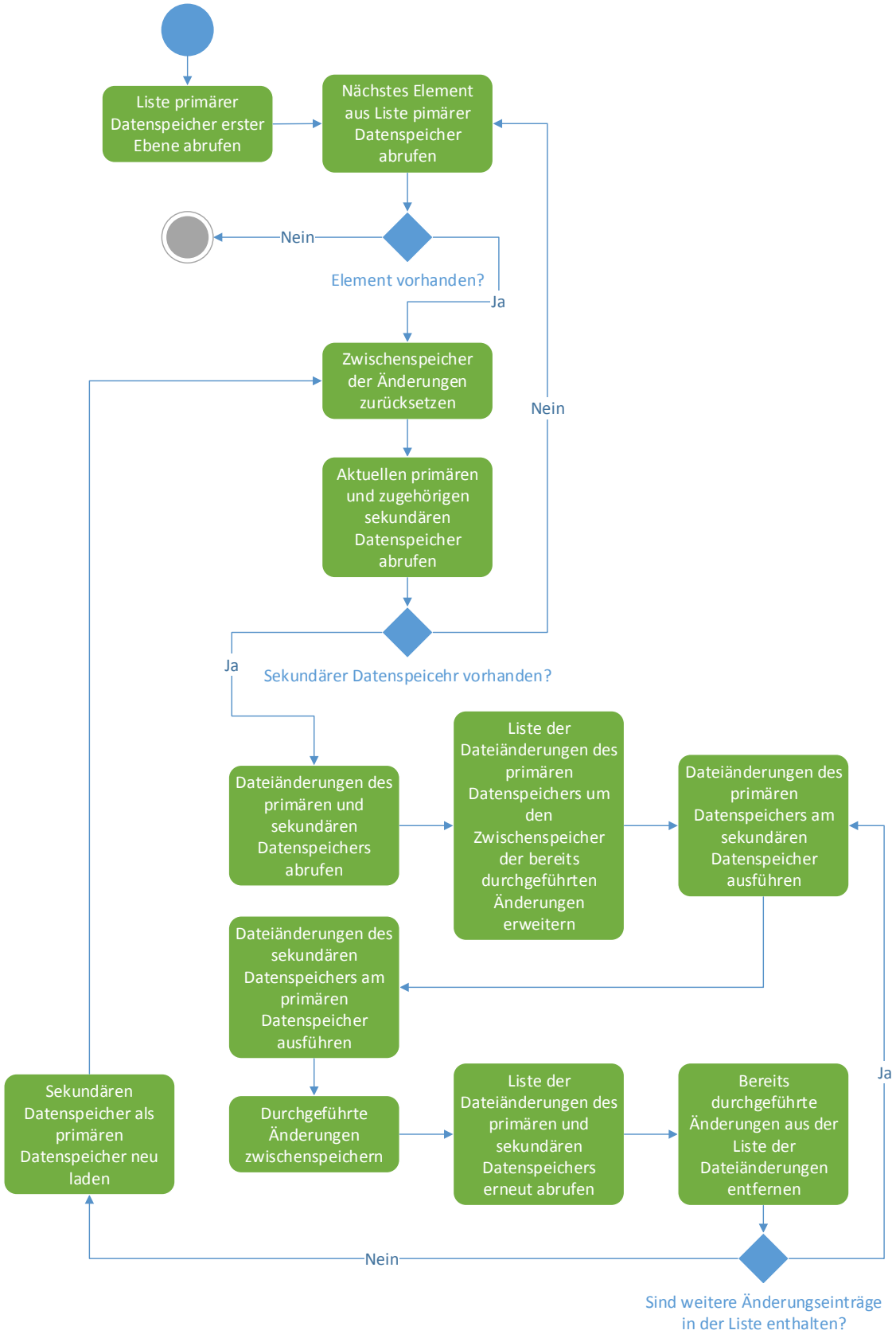

Abbildung 27 - Synchronisation der virtuellen Datenspeicher

4.5 Verschlüsselung

Damit die Daten einzelner virtueller Verzeichnisse verschlüsselt auf den externen Datenspeichern abgelegt werden können, werden in der Klasse Datastorage die Funktionen csp_encryptStream und csp_decryptStream implementiert, die den Filestream der Uploads verschlüsseln und den der Downloads entschlüsseln. Die durchgeführten Kryptographie Methoden werden damit von den Anwendern unbemerkt angewendet und müssen von diesen auch nicht aktiv anstoßen werden.

Wenn eine Verschlüsselung durchgeführt werden soll, muss lediglich ein Wert in der Klassenvariable krypto_key gesetzt sein. Die beiden Kryptographiefunktionen werden dann automatisch in den Funktionen getFile, uploadFile und uploadFileStream zum ver- beziehungsweise entschlüsseln des jeweiligen Filestreams aufgerufen.

Die Funktionen csp_encryptStream und csp_decryptStream nutzen dafür die PHP Funktion stream_filter_append, welche unterschiedliche Filteroperationen an Streams durchführen kann. Die Implementierung im CSP nutzt dabei eine 3DES (Triple Data Encryption Standard) Stream Verschlüsselung, welche mit einem Initialisierungsvektor und einem Schlüssel arbeitet. Diese werden wiederum mit der in der Variable krypto_key übergebenen Passphrase generiert.

Sollte für einzelne Implementierungen der Klasse Datastorage anderer Kryptographie Verfahren gewünscht sein, können diese einfach durch Überlagerung der beiden Funktionen ausgetauscht werden.

Der Sourcecode der Implementierung der Ver- und Entschlüsselungsmethoden befindet sich in der Klasse Datastorage im Anhang D.9.

4.6 Weboberfläche

Um dem Anwender einen direkten Zugriff auf die virtuelle Ordnerstruktur des CSP zu ermöglichen, wurde eine eigene Weboberfläche entwickelt. Ziel war es dabei, dem Anwender eine Interaktion mit der virtuellen Ordnerstruktur zu ermöglichen und alle Operationen innerhalb der externen Datenspeicher einheitlich durchzuführen, ohne das es für den Anwender relevant ist, in welchem externen Datenspeicher er sich gerade befindet.

Die gesamte Weboberfläche wurde in zwei Anwendungsdateien abgebildet:

- index.php Abarbeitung aller Anwenderinteraktionen und
- getfile.php Download einer Datei.

Die Anwendung arbeitet dabei durchgängig mit den Übergabeparametern

- ds_id ID des aktuell geöffneten virtuellen Ordners,
- ds_path Aktuell zu öffnender Pfad innerhalb des virtuellen Ordners und
- ds_delete optional zu löschende Datei oder zu löschenden Ordner innerhalb von ds_path.

Werden keine Parameter übergeben, wird die virtuelle Ordnerstruktur angezeigt (siehe Abbildung 28).

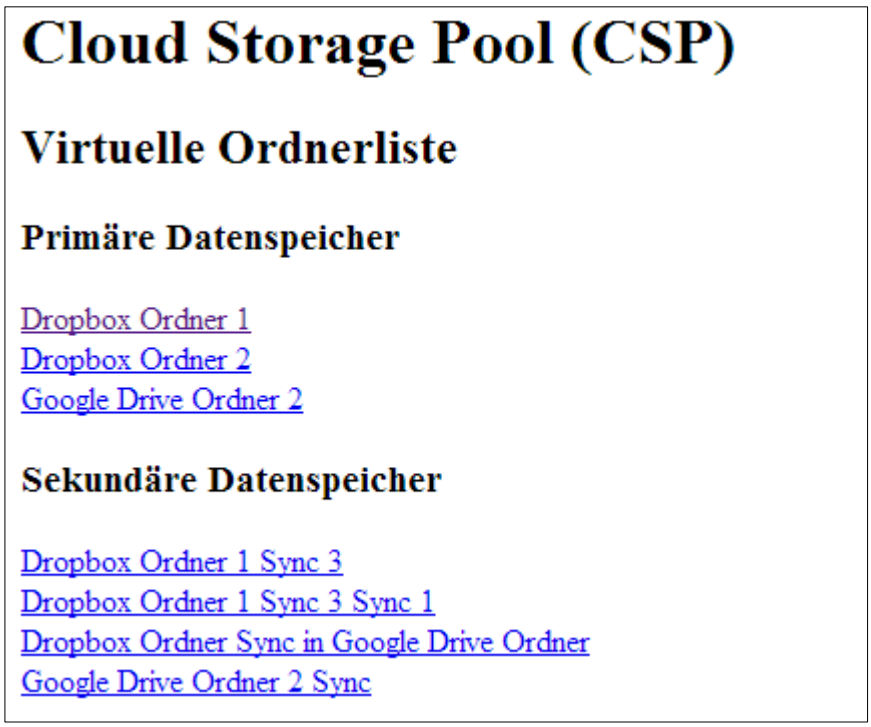

Abbildung 28 - Virtuelle Ordnerstruktur in der Weboberfläche

Zusätzlich gibt es Parameter, welche mittels HTTP Post übergeben werden können, wenn ein Dateiupload oder das Anlegen eines neuen Ordners durchgeführt werden soll:

- userfile Postvariable vom Typ File mit der hochzuladenden Datei und
- newfolder Name des neu anzulegenden Ordners.

In der Benutzeransicht wird in allen Listen zwischen Ordnern und Dateien unterschieden. Der Anwender kann durch Ordner browsen und sie löschen. Dateien können geöffnet und ebenfalls gelöscht werden. Für das Öffnen einer Datei wird die Programmdatei getfile.php aufgerufen, welche den Dateiinhalt ausgibt und auch die korrekten Headerinformationen an den Client sendet.

Die Ansicht des Inhaltes eines virtuellen Ordners ist in Abbildung 29 ersichtlich.

Abbildung 29 - Inhalt eines virtuellen Ordners in der Weboberfläche

Der Sourcecode der Implementierung der Weboberfläche befindet sich in den Anhängen D.5. und D.6.

4.7 Clientzugriff über WebDAV

Für den direkten Zugriff von Clientanwendungen und der Einbindung in das Dateisystem von Endgeräten wurde eine WebDAV Schnittstelle entwickelt. Es handelt sich dabei um eine Implementierung der Klasse HTTP_WebDAV_Server von Christian Stocker und Hartmut Holzgraefe, welche als Teil von Pear [Hol12] verfügbar ist.

Die Klasse HTTP_WebDAV_Server hat dabei die gesamte Verarbeitung der Serveranfragen bereits implementiert. In der eigenen Ableitung der Klasse müssen dann jeweils die Methoden

- checkAuth
- PROPFIND
- GET
- GetDir
- PUT
- MKCOL
- DELETE

- MOVE
- COPY und
- PROPPATCH

des WebDAV Servers implementiert werden.

Die Implementierung erfolgte für den Zugriff auf den CSP mit der Klasse CspWebdav. Der Aufruf des Servers erfolgt dann direkt über die Programmdatei webdav.php, welche die Instanz der Klasse CspWebdav anlegt, die Konfigurationsdaten des CSP mittels Aufruf der Funktion *setConfigAppinfos* initialisiert und danach die Abarbeitung des WebDAV Request durch Aufruf der Funktion *ServeRequest* anstößt.

Das Klassendiagramm der Klasse CspWebdav ist in Abbildung 30 dargestellt.

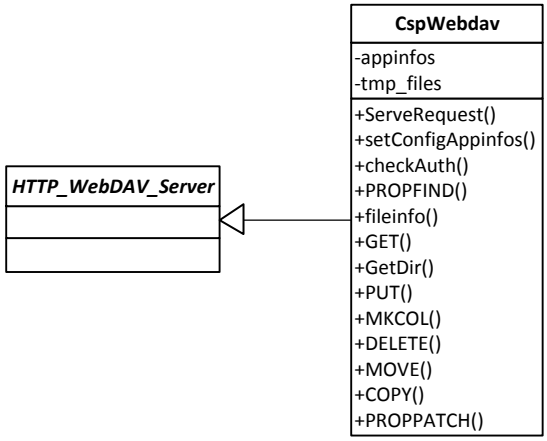

Abbildung 30 - Klassendiagramm CspWebdav

Für den Zugriff auf den WebDAV Server gibt es eine Vielzahl von Clientapplikationen. Beispiele dafür sind CarrotDAV [Hob14] und Webdrive [Sou14] im PC Bereich und die mobile Anwendung Woopiti [Alv14] für Zugriffe über das Smartphone.

Die Zugriffe über CarrotDAV, Webdrive und Woopiti sind nachfolgend in Abbildung 31, Abbildung 32 und Abbildung 33 dargestellt. Der Sourcecode der Implementierung der WebDAV Schnittstelle befindet sich in den Anhängen D.8. und D.12.

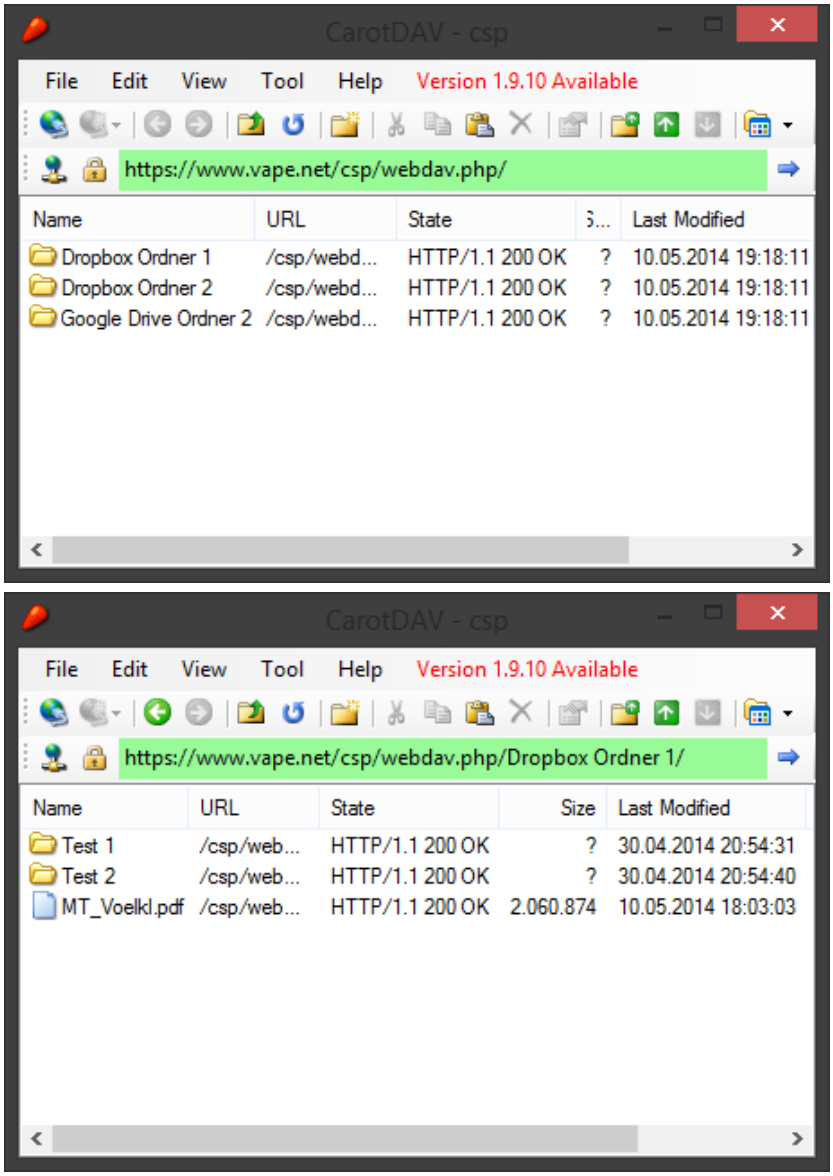

Abbildung 31 - WebDAV Zugriff auf den CSP mittels CarotDAV

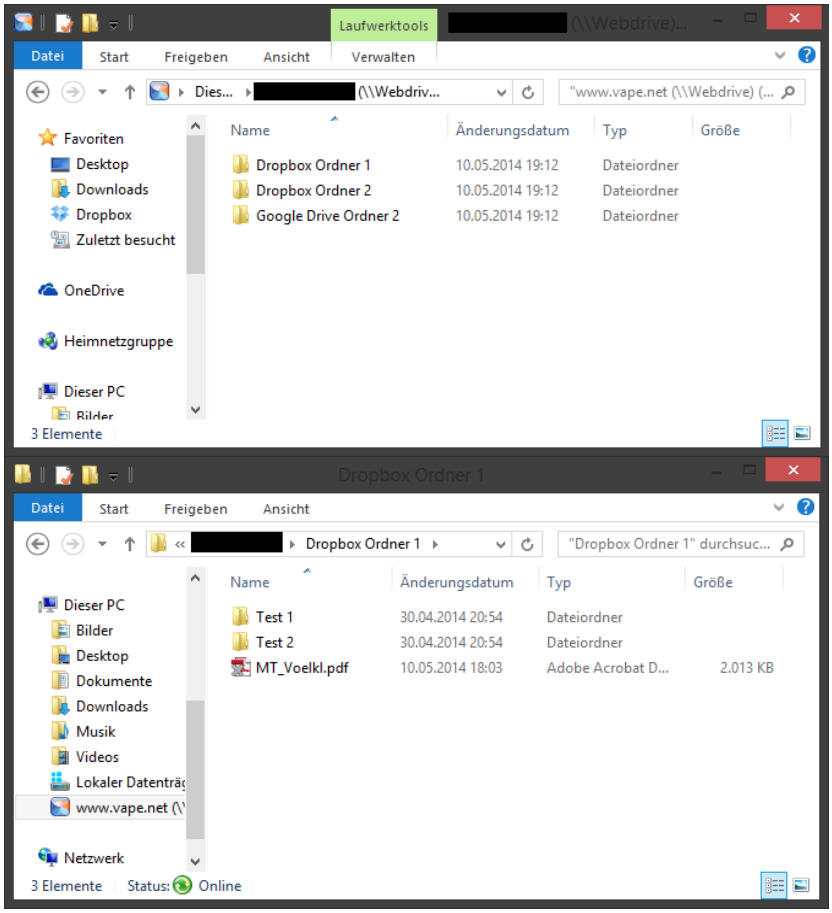

Abbildung 32 - WebDAV Zugriff auf den CSP mittels Webdrive

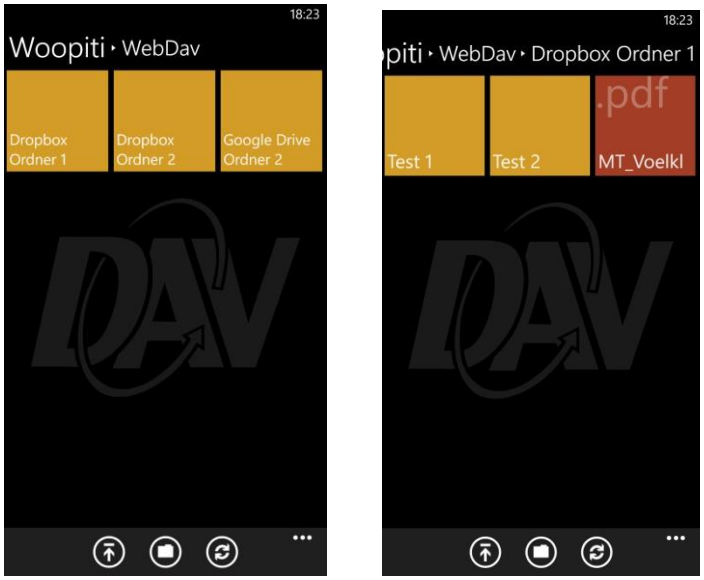

Abbildung 33 - WebDAV Zugriff auf den CSP mittels Woopiti

5 Schlussfolgerungen und Ausblick

Diese Master These beschäftigte sich mit Cloud Speicherdiensten und ihrer virtuellen Vernetzung und gliederte sich dabei in fünf Abschnitte. Im ersten Abschnitt wurde die Problemstellung und Zielsetzung definiert und der Aufbau der Arbeit festgelegt. Der nachfolgende Abschnitt beschäftigte sich mit den Grundlagen des Cloudcomputing, wobei hier auf seine Entstehung und die einzelnen Servicearten HuaaS, SaaS, PaaS und IaaS eingegangen wurde. Im Detail wurde dabei der Bereich der Cloud Storage Services und der zugehörigen Cloud Storage Security behandelt. Das darauf folgende Kapitel 3, Related Work, beschäftigte sich mit den technischen Grundlagen der Datenschnittstellen SOAP, REST und WebDAV, sowie den unterschiedlichen, bei den Endanwendern am häufigsten genutzten, Cloud Storage Services und den zugehörigen APIs. Weiter wurden auch bereits existierende Lösungen betrachtet und verglichen, welche sich mit der Integration und Vernetzung von Cloud Speichern beschäftigen. Daraus wurde dann auch die Notwendigkeit einer eigenen Entwicklung abgeleitet. Im anschließenden Abschnitt wurde die Implementierung des Prototyps inklusive der Anbindung an Dropbox und Google Drive geplant, spezifiziert und durchgeführt.

Eine verteilte Speicherung von Daten in mehreren Cloud Storages ist möglich. Die entwickelte Zwischenschicht „Cloud Storage Pool (CSP)“ deckt dabei alle Anforderungen ab, die notwendig sind eine Vernetzung so durchzuführen, um dem Anwender einen zentralen Zugriff auf mehrere unterschiedliche Datenspeicher zu ermöglichen. Dies sind insbesondere die transparente und integrierte Darstellung der Ordnerstruktur und die einheitliche Ablage und Abrufung von Dateien und Ordnern. Durch die Implementierung einer zentralen abstrakten Klasse mit einheitlichen Methoden und Strukturen kann eine leichte Erweiterbarkeit um jeden beliebigen Datenspeicherservice sichergestellt werden. Bei einzelnen Anbietern, wie Google Drive, ist es notwendig, den Abruf von Pfad-basierten Zugriffen nachträglich zu erweitern, sofern diese keinen direkten Pfadzugriff unterstützen. Auch müsste bei Anbietern ohne Unterstützung des Abrufes von Dateiänderungen diese Funktionalität zusätzlich implementiert werden.

Eine Stream Ver- und Entschlüsselung der Dateiübertragungen ermöglicht eine sichere Ablage von vertraulichen Informationen in der Cloud, ohne einen zusätzlichen Aufwand für den Anwender zu verursachen. Auch ist es möglich die Ver- und Entschlüsselung für jeden externen Speicherort separat zu aktivieren oder zu deaktivieren und mit jeweils eigenen Schlüsseln zu versehen.

Die Forschungsfrage und These: „**Es ist möglich, eine betriebssystemunabhängige Webanwendung zu entwickeln, die als Zwischenschicht zwischen Anwender und mehreren Cloud Storage Services arbeitet und die redundante und verschlüsselte Speicherung von in der Cloud gespeicherten Daten erlaubt, so dass der Anwender**

über eine virtuelle Dateistruktur direkt, und ohne der Notwendigkeit des Wissens über den dahinterliegenden Speicherort, darauf zugreifen kann.“ kann somit als gültig bestätigt werden, da die Entwicklung einer Anwendung mit den genannten Anforderungen möglich ist.

Zukünftig ist eine Weiterentwicklung des CSP Prototypen um eine Benutzerverwaltung, eine erweitertes Userinterface mit Konfigurationsmöglichkeiten und die Integration von lokalen Datenspeichern anzustreben. Weiter ist eine Integration in Portalanwendungen und lokale Netzwerkgeräte wie Network Attached Storages (NAS) und All-in-One Router denkbar, um Endanwendern einen eigenen und leicht zu konfigurierenden lokalen Betrieb des CSP zu ermöglichen. Die daraus entstehenden Anwendungsmöglichkeiten sind sehr vielfältig, da es sich um eine neue Zwischenschicht handelt, welche in bestehende Speicherzugriffe integriert werden kann. Somit wäre auch eine Konstellation denkbar, in der die Anwendungsdateien und Daten eines Webservers verschlüsselt und auf mehrere unabhängige Cloudspeicher abgelegt sind. Der Zugriff des Webservers erfolgt dabei direkt über WebDAV. Auch ein Einsatz im Umfeld von Businessanwendern ist denkbar. So wäre es möglich, den Mitarbeitern einen kleinen CSP Webserver zur Verfügung zu stellen, welcher ihnen den Zugriff auf ihre Daten ermöglicht, ohne dabei eine direkte Verbindung ins Unternehmensnetzwerk zu benötigen. Die Daten könnten dabei vom unternehmenseigenen Dateiserver über eine Synchronisation durch einen CSP Service verschlüsselt mit einem Cloudspeicher abgeglichen werden, welcher dann wiederum einen dezentralen Zugriff über den mobilen CSP Server des Mitarbeiters ermöglicht. Damit bleiben die zukünftig möglichen Erweiterungen nicht nur auf Anwendungsfälle für Endanwender beschränkt, sondern es ergibt sich ergänzend auch eine Vielzahl von Integrationsmöglichkeiten im Businessumfeld.

Literaturverzeichnis

[Ado14] Adobe Systems Incorporated (2014): *Photoshop Online Tools*. http://www.photoshop.com/tools, zuletzt geprüft am 15.April.2014.

[Aka14] Akamai Technologies (2014): *Akamai EdgePlatform Products*. http://www.akamai.com/html/technology/products/, zuletzt geprüft am 15.April.2014.

[Alv14] Alvaro Rivoir: *Woopiti | Windows Phone Apps+Games Store*. http://www.windowsphone.com/en-us/store/app/woopiti/20e820bf-56a3-4ec5-a839-3cb2886c64b6, zuletzt geprüft am 10.Mai.2014.

[Ama14] Amazon Web Services, Inc. (2014): *AWS | Amazon Elastic Compute Cloud (EC2)*. https://aws.amazon.com/de/ec2/, zuletzt geprüft am 15.April.2014.

[Ama141] Amazon Web Services, Inc. (2014): *AWS | Amazon Simple Storage Service (S3)*. https://aws.amazon.com/de/s3/, zuletzt geprüft am 15.April.2014.

[Ama142] Amazon Web Services, Inc. (2014): *AWS | Amazon SimpleDB*. http://aws.amazon.com/de/simpledb/, zuletzt geprüft am 15.April.2014.

[Ame14] Amer S.; Dittrich I. und Fink S.: *Cloud Computing via Dropbox - Geschichte Dropbox*. http://dropboxinfo.npage.de/geschichte-dropbox.html, zuletzt geprüft am 16.April.2014.

[Apa14] Apache Software Foundation (2014): *HDFS Architecture Guide*. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, zuletzt geprüft am 1.Juni.2014.

[Arm09] Armbrust M.; Fox A.; Griffith R. et al. (2009): *Above the Clouds: A Berkeley View of Cloud Computing, Technical Report No. UCB/EECS-2009-28*. Berkeley: Electrical Engineering and Computer Sciences University of California at Berkeley.

[Bab14] Babiarz P. (2014): *Sicherheitsauswirkungen von verknüpften Cloud-Diensten*. Wien: Fachhochschule Technikum Wien.

[Bau11] Baun C.; Kunze M.; Nimis J. et al. (2011): *Cloud Computing, Web-basierte dynamische IT-Services*. Berlin Heidelberg: Springer Verlag.

[Blu14] Bluelock (2014): *BlueLock Cloud Services*. http://www.bluelock.com/cloud-services/, zuletzt geprüft am 15.April.2014.

[Cha06] Chang F.; Dean J.; Ghemawat S. et al. (2006): *Bigtable: A Distributed Storage System for Structured Data*. Seattle, WA: OSDI'06: Seventh Symposium on Operating System Design and Implementation.

[Clo14] CloudKafe: *CloudKafe - Organizing your Cloud!* https://www.cloudkafe.com/, zuletzt geprüft am 18.April.2014.

[Dro14] Dropbox (2014): *Dropbox*. https://www.dropbox.com/, zuletzt geprüft am 15.April.2014.

[Dro141] Dropbox: *Dropbox - Developers*. https://www.dropbox.com/developers, zuletzt geprüft am 16.April.2014.

[Dro142] Dropbox: *What compliance documentation or certifications does Dropbox have?* https://www.dropbox.com/help/238/en, zuletzt geprüft am 17.April.2014.

[Dro143] Dropbox: *Core API*. https://www.dropbox.com/developers/core/api, zuletzt geprüft am 18.April.2014.

[Dro144] Dropbox Inc.: *PHP Core SDK*. https://www.dropbox.com/developers/core/sdks/php, zuletzt geprüft am 18.April.2014.

[Dro145] Dropbox: *App Console - Dropbox*. https://www.dropbox.com/developers/apps, zuletzt geprüft am 10.Mai.2014.

[Eur00] Europäische Union (2000): *EUR-Lex - 32000D0520 - DE, Amtsblatt Nr. L 215 vom 25/08/2000 S. 0007 - 0047*. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32000D0520:de:HTML, zuletzt geprüft am 17.April.2014.

[Eur001] Europäische Union (2000): *Entscheidung der Kommission vom 26. Juli 2000 gemäß der Richtlinie 95/46/EG des Europäischen Parlaments und des Rates*. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2000:215:0007:0047:DE:PDF, zuletzt geprüft am 18.April.2014.

[Eur12] EuroCloud.Austria; Wirtschaftskammer Wien; Austrian Standards Institute (2012): *Cloud-Verträge, Was Anbieter und Kunden Besprechen sollten*. 1. Aufl. Wien: EuroCloud.Austria.

[Fac14] Facebook (2014): *Facebook-Entwickler*. https://developers.facebook.com, zuletzt geprüft am 15.April.2014.

[Fie00] Fielding R.T. (2000): *Architectural Styles and the Design of Network-based Software Architectures, CHAPTER 5, Representational State Transfer (REST)*. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, zuletzt geprüft am 1.Juni.2014.

[flu14] fluid Operations AG (2014): *eCloudManager*. http://www.fluidops.com/ecloudmanager-de/?lang=de, zuletzt geprüft am 15.April.2014.

[Ghe03] Ghemawat S.; Gobioff H. und Leung S.-T. (2003): *The Google File System*. Lake George, NY: 19th ACM Symposium on Operating Systems Principles.

[GoG14] GoGrid (2014): *GoGrid's Cloud Platform*. http://www.gogrid.com/cloud-platform, zuletzt geprüft am 15.April.2014.

[Goo11] Google Inc. (2011): *Our commitment to the Safe Harbor privacy framework*. http://googleenterprise.blogspot.co.at/2011/09/our-commitment-to-safe-harbor-privacy.html, zuletzt geprüft am 17.April.2014.

[Goo14] Google Inc. (2014): *Google Docs*. http://docs.google.com/?hl=de, zuletzt geprüft am 15.April.2014.

[Goo141] Google Inc. (2014): *Google Maps API*. https://developers.google.com/maps/?hl=de-DE, zuletzt geprüft am 15.April.2014.

[Goo142] Google Inc. (2014): *OpenSocial*. https://developers.google.com/opensocial/?hl=de, zuletzt geprüft am 15.April.2014.

[Goo143] Google Inc. (2014): *Google App Engine*. https://appengine.google.com/, zuletzt geprüft am 15.April.2014.

[Goo144] Google Inc. (2014): *Google Drive*. https://drive.google.com, zuletzt geprüft am 16.April.2014.

[Goo145] Google Inc.: *Google Drive SDK - API Reference*. https://developers.google.com/drive/v2/reference/, zuletzt geprüft am 18.April.2014.

[Goo146] Google Inc.: *Google APIs Client Library for PHP*. https://code.google.com/p/google-api-php-client/, zuletzt geprüft am 18.April.2014.

[Goo147] Google Inc.: *Google Developers Console*. https://console.developers.google.com, zuletzt geprüft am 10.Mail.2014.

[Gug13] Guger J. (2013): *Outsourcing, Offshoring and Alliances*. Wien: FFH Gesellschaft zur Erhaltung und Durchführung von Fachhochschulstudiengängen m. b. H.

[Gus02] Alonso G.; Casati F.; Kuno H. et al. (2002): *Web Services - Concepts, Architectures and Applications [Kindle Edition]*. Hong Kong: Springer Verlag.

[Hau13] Hausdorf M. (2013): *Cloud-Speicher sicher und effizient nutzen*. Wien: Fachhochschule Technikum Wien.

[Hei12] Heise Zeitschriften Verlag (2012): *Google öffnet seinen Cloud-Speicher*. http://www.heise.de/newsticker/meldung/Google-oeffnet-seinen-Cloud-Speicher-1557903.html, zuletzt geprüft am 17.April.2014.

[Hob14] Hobara, Rei: *WebDAV Client CarotDAV*. http://rei.to/carotdav_en.html, zuletzt geprüft am 10.Mai.2014.

[Hol12] Holzgraefe H. und Stocker C. (2012): *HTTP_WebDAV_Server*. The PHP Group. http://pear.php.net/package/HTTP_WebDAV_Server/redirected, zuletzt geprüft am 10.Mai.2014.

[Int09] International Telecommunication Union (2009): *H.323 : Packet-based multimedia communications systems*. http://www.itu.int/rec/T-REC-H.323-200912-I/en, zuletzt geprüft am 1.Juni.2014.

[Jol14] Jolicloud: *Jolicloud*. http://www.jolicloud.com/, zuletzt geprüft am 18.April.2014.

[JuW11] Ju J.; Wu J.; Fu J. et al. (2011): *A Survey on Cloud Storage*. JOURNAL OF COMPUTERS, VOL. 6, NO. 8, AUGUST 2011.

[Kav13] Kavitha M.G.; Vinay Kumar N.A. und Balasubrhamanya (2013): *Secure Cloud Storage with Multi Cloud Architecture*. International Journal of Innovative Technology and Exploring Engineering, Vol 3, Iss 1, Pp 45-49.

[Mes14] Messina C. und Parecki A.: *OAuth Community Site*. http://oauth.net/, zuletzt geprüft am 18.April.2014.

[Mic13] Microsoft Corporation (2013): *Onlinedatenschutzbestimmungen von Microsoft*. http://privacy.microsoft.com/DE-AT/fullnotice.mspx, zuletzt geprüft am 17.April.2014.

[Mic141] Microsoft Corporation (2014): *Microsoft Office Online*. https://office.com/start/default.aspx, zuletzt geprüft am 15.April.2014.

[Mic142] Microsoft Corporation (2014): *Microsoft Azure*. http://www.azure.microsoft.com/en-us/, zuletzt geprüft am 15.April.2014.

[Mic143] Microsoft Corporation (2014): *OneDrive*. https://onedrive.live.com/, zuletzt geprüft am 15.April.2014.

[Mic144] Microsoft Corporation: *OneDrive API*. http://msdn.microsoft.com/en-us/library/windows/apps/hh826521.aspx, zuletzt geprüft am 18.April.2014.

[Mic145] Microsoft Corporation (2014): *[MS-SMB2]: Server Message Block (SMB) Protocol Versions 2 and 3*. http://msdn.microsoft.com/en-us/library/cc246482.aspx, zuletzt geprüft am 1.Juni.2014.

[Mic146] Microsoft Corporation (2014): *[MS-CIFS]: Common Internet File System (CIFS) Protocol*. http://msdn.microsoft.com/en-us/library/ee442092.aspx, zuletzt geprüft am 1.Juni.2014.

[Mul11] Mulazzani M.; Schrittwieser S.; Leithner M. et al. (2011): *Cloud Speicherdienste als Angriffsvektoren*. Wien: 9th Information Security Konferenz in Krems.

[Nat11] National Institute of Standards and Technology (2011): *The NIST Definition of Cloud Computing*. Gaithersburg.

[Net14] NetSuite Inc. (2014): *Suiteflex*. http://www.netsuite.co.uk/portal/uk/developers/suiteflex.shtml, zuletzt geprüft am 15.April.2014.

[Net141] NetMediaEurope Deutschland GmbH (2014): *Microsoft SkyDrive heißt ab sofort OneDrive und bietet neue Funktionen*. http://www.zdnet.de/88184613/microsoft-skydrive-heisst-ab-sofort-onedrive-und-bietet-neue-funktionen/, zuletzt geprüft am 17.April.2014.

[Ope14] OpenID (2014): *OpenID Foundation*. http://openid.net/foundation/, zuletzt geprüft am 15.April.2014.

[Own14] ownCloud Inc.: *ownCloud*. http://owncloud.org/, zuletzt geprüft am 18.April.2014.

[Own141] ownCloud Inc.: *ownCloud Administrators Manual*. http://doc.owncloud.org/server/6.0/admin_manual/, zuletzt geprüft am 18.April.2014.

[Pad12] Padhy R.P. und Patra M.R. (2012): *Evolution of Cloud Computing and Enabling Technologies*. International Journal of Cloud Computing and Services Science (IJ-CLOSER), Vol.1, No.4, October 2012, pp. 182-198.

[QSu14] Q-Success (2014): *Usage Statistics and Market Share of Server-side Programming Languages for Websites, May 2014*. http://w3techs.com/technologies/overview/programming_language/all, zuletzt geprüft am 09.05.2014.

[Rac14] Rackspace (2014): *SSD Cloud Server und Cloud Server Hosting von Rackspace*. http://www.rackspace.com/de/cloud/servers, zuletzt geprüft am 15.April.2014.

[RFC02] RFC Network Working Group (2002): *RFC 3261, SIP: Session Initiation Protocol*. http://tools.ietf.org/html/rfc3261, zuletzt geprüft am 1.Juni.2014.

[RFC03] RFC Network Working Group (2003): *RFC 3530, Network File System (NFS) version 4 Protocol*. http://tools.ietf.org/html/rfc3530, zuletzt geprüft am 1.Juni.2014.

[RFC08] RFC Network Working Group (2008): *RFC 5321, Simple Mail Transfer Protocol*. http://tools.ietf.org/html/rfc5321, zuletzt geprüft am 1.Juni.2014.

[RFC99] RFC Network Working Group (1999): *RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1*. http://tools.ietf.org/html/rfc2616, zuletzt geprüft am 1.Juni.2014.

[RFC991] RFC Network Working Group (1999): *RFC 2617, HTTP Authentication: Basic and Digest Access Authentication*. http://tools.ietf.org/html/rfc2617, zuletzt geprüft am 1.Juni.2014.

[sal14] salesforce.com, inc. (2014): *CRM Software & Online CRM System - Salesforce.com Deutschland*. http://www.salesforce.com/de/, zuletzt geprüft am 15.April.2014.

[sal141] salesforce.com Inc. (2014): *What is Force.com*. http://www.salesforce.com/platform/what/, zuletzt geprüft am 15.April.2014.

[Sch12] Schroder C. (2012): *How To Synchronize Dropbox and ownCloud on Linux*. Linux.com. https://www.linux.com/learn/tutorials/647781-how-to-synchronize-dropbox-and-owncloud-on-linux, zuletzt geprüft am 18.April.2014.

[Sky14] Skytap, Inc. (2014): *Enterprise Dev/Test Environments*. http://www.skytap.com/solutions/development-and-test, zuletzt geprüft am 15.April.2014.

[Sou14] South River Technologies: *WebDrive FTP Client*. http://www.webdrive.com/product-downloads/, zuletzt geprüft am 10.Mai.2014.

[Spi12] Spiegel Online GmbH (2012): *Gratis-Onlinespeicher SkyDrive: Microsoft öffnet seine eigene Dropbox*. http://www.spiegel.de/netzwelt/web/skydrive-microsofts-cloud-speicher-mit-mehr-funktionen-a-829371.html, zuletzt geprüft am 17.April.2014.

[Til11] Tilkov S. (2011): *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien [Kindle Edition]*. 2. akt. und erw. Auflage. Aufl. Heidelberg: Dpunkt Verlag.

[USH14] US Handelsministrium: *Safe Harbor - List*. http://safeharbor.export.gov/list.aspx, zuletzt geprüft am 18.April.2014.

[W3C06] W3C (2006): *Extensible Markup Language (XML) 1.1 (Second Edition)*. http://www.w3.org/TR/2006/REC-xml11-20060816/, zuletzt geprüft am 1.Juni.2014.

[W3C07] W3C (2007): *SOAP Specifications*. http://www.w3.org/TR/soap/, zuletzt geprüft am 1.Juni.2014.

[Wag14] Wagner A. (2014): *Privacy in Google-based Services*. Wien: Fachhochschule Technikum Wien.

[Zoh14] Zoho Corporation Pvt. Ltd. (2014): *Online database apps in minutes*. https://www.zoho.com/creator, zuletzt geprüft am 15.April.2014.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

3DES	Triple Data Encryption Standard
API	Application Programming Interface
CIFS	Common Internet File System
CSP	Cloud Storage Pool
CSV	Comma Seperated Value
DaaS	Database as a Service
DNS	Domain Name System
DSG2000	Datenschutzgesetz 2000
ECOCloudS	Efficient Combination Of Cloud Storages
ER Modell	Entity Relationship Modell
GFS	Google File System
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HTTP	Hyper Text Transfer Protokoll
HTTP	Hyper Text Transfer Protokoll
HuaaS	Humans as a Service
IaaS	Infrastructure as a Service
NAS	Network Attached Storage
NFS	Network File System
NIST	National Institute of Standards and Technologie
PaaS	Platform as a Service
PC	Personal Computing ()
REST	Representational State Transfer
SaaS	Software as a Service
SHA	Secure Hash Algorithm

SMB	Server Message Block
SMTP	Simple Mail Transport Protokoll
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
VoIP	Voice oder Internet Protikoll
WebDAV	Webbased Distributed Authoring and Versioning
WWW	World Wide Web
XML	Extensible Markup Language

Anhang A: Dropbox API Methoden

Nachfolgend sind die wichtigsten Befehle der Dropbox API aufgelistet und im Detail beschrieben. [Dro143]

Alle Befehle werden durch HTTP Methoden ausgeführt.

GET /files/<root>/path
Rut die Dateiliste eines Ordners oder den Inhalt einer Datei, inklusive der zugehörigen Metadaten, ab. Der Rückgabewert ist in einer JSON Struktur abgebildet und beinhaltet die Parameter:

- size — Größe der Datei in einem User lesbaren Format,
- bytes — Größe der Datei in Byte,
- path — Pfad zur Datei,
- is_dir — Gibt an, ob es sich um einen Ordner handelt, oder nicht,
- is_deleted — Gibt an, ob es sich um eine bereits gelöschte Datei handelt,
- rev — Aktuelle Revisionsnummer der Datei,
- hash — Hashwert der Datei,
- thumb_exists — Gibt an, ob die Datei als Thumbnail abgerufen werden kann,
- photo_info — Medieninfo, wenn es sich um eine Grafikdatei handelt,
- video_info — Medieninfo, wenn es sich um eine Videodatei handelt,
- icon — Verweis auf das zugehörige Icon der Dropbox Icon Bibliothek,
- modified — Datum und Uhrzeit der letzten Änderung,
- client_mtime — MIME Typ der Datei,
- root — Root Ordner der Datei und
- revision — Depricated Version des Feldes rev.

POST /files_put/<root>/<path>
Hochladen einer Datei in die Dropbox. Die hochzuladende Datei befindet sich dabei im Request Body. Mit dem Parameter *overwrite* kann ein automatisches Überschreiben einer bestehenden Datei angefordert werden. Anderenfalls behält die Dropbox beide Dateien und ergänzt die zusätzlichen Dateien um eine automatische Nummerierung.

Als Rückgabewert wird eine JSON Struktur mit den Metadaten der Datei zurückgeliefert (siehe GET /files/<root>/path).

GET /metadata/<root>/<path>
Abruf der Metadaten eines Ordners oder einer Datei in Form einer JSON Struktur (siehe GET /files/<root>/path). Gelöschte Dateien werden automatisch ausgebendet. Sollte ihre Anzeige gewünscht sein, muss zusätzlich der Parameter *include_deleted* gesetzt werden.

GET /delta
Abfrage der geänderten Daten verglichen mit dem letzten bekannten Stand der Anwendung zurück. Der letzte bekannte Stand wird durch einen Cursor definiert, der an den Aufruf im Parameter *cursor* übergeben wird. Ohne Übergabe eines Cursors werden alle Änderungen seit Bestehen des Datenspeichers zurückgeliefert. Die Rückgabewerte beinhalten wiederum den nächsten Cursorwert, ab dem wiederum abgefragt werden kann.

Der Rückgabewert entspricht einer JSON Struktur mit den Elementen:

- entries Liste von Delta Einträgen,
- reset ob es sich um eine Abfrage mit, oder ohne bestehenden Cursor gehandelt hat,
- cursor nächster Cursor der Abfrage und
- has_more wenn die Abfragemenge an Deltaeinträgen beschränkt war, wird hier angegeben, ob weitere Elemente abrufbar sind.

Die Delta Einträge sind innerhalb oben genannten JSON Struktur im Eintrag entries abgebildet und haben den Aufbau:

- pfad Pfad zum geänderten Element und
- metadata Metadaten des geänderten Elements (null, wenn das Element gelöscht wurde)

POST /fileops/copy
Kopiert einen Ordner oder eine Datei innerhalb der Dropbox. Die verpflichtenden Parameter sind:

- root Rootverzeichnis ab dem der from_path und to_path gelten,
- from_path Pfad der Quelldatei und
- to_path Pfad der Zieldatei.

Als Rückgabewert werden die Metadaten der kopierten Datei zurückgeliefert.

POST /fileops/create_folder
Erstellt einen neuen Ordner innerhalb der Dropbox. Als verpflichtende Übergabewerte werden

- root Rootverzeichnis des neuen Ordners und
- path Pfad des neuen Ordners ab root

übergeben.

Als Rückgabewert werden die Metadaten des neuen Ordners zurückgeliefert.

POST /fileops/delete
Löscht einen Ordner oder eine Datei. Als verpflichtende Übergabewerte werden

- root Rootverzeichnis des zu löschenden Ordners oder der Datei und
- path Pfad des zu löschenden Ordners oder der Datei ab root

übergeben.

Als Rückgabewert werden die Metadaten des gelöschten Objekts zurückgeliefert.

POST /fileops/move
Verschiebt einen Ordner oder eine Datei. Die verpflichtenden Parameter sind:

- root Rootverzeichnis ab dem der from_path und to_path gelten,
- from_path Pfad der Quelldatei und
- to_path Pfad der Zieldatei.

Als Rückgabewert werden die Metadaten des verschobenen Objekts zurückgeliefert.

GET /account/info
Liefert Informationen zum Dropbox Account in Form einer JSON Struktur zurück. Der Aufbau der Rückgabestruktur ist in Tabelle 6 in Kapitel 3.3.1 beschrieben.

GET /oauth2/authorize
Startet einen neuen Autorisierungsworkflow für einen neuen Dropbox Account. Es wird eine Webseite ausgegeben, die dem Anwender die Verknüpfung der Dropbox Anwendung mit seinem Account ermöglicht.

Die verpflichtenden Übergabewerte sind die *client_id* der Applikation und der *response_type* der Anfrage. Mögliche *response_type* Werte sind *token* und *code*.

Der Code kann verwendet werden um in der Anwendung einen Token anzufordern. Er ist nur temporär gültig. Ein Token wird zur langfristigen Verknüpfung der Anwendung mit dem Account benötigt. Er ermöglicht auch eine Offline-Verbindung der Anwendung ohne aktive Interaktion durch den Anwender.

POST /oauth2/token
Liefert einen Access Token für einen bereits verbundenen Dropbox Account zurück. Dadurch wird der verbundenen Anwendung ermöglicht einen Code (siehe *GET /oauth2/authorize*) in einen Token umzuwandeln.

Die verpflichtenden Parameter des Aufrufs sind *code* und *grant_type*, wobei *grant_type* immer den Wert *authorisation_code* hat.

POST /disable_access_token
Deaktiviert einen gültigen Access Token, um die Verknüpfung zwischen Anwendung und Dropbox Account zu löschen. Ein deaktivierter Token ist nicht mehr verwendbar und muss für eine erneute Verknüpfung neu angefordert werden.

Anhang B: HTTP Error Codes

Nachfolgend sind alle standardmäßig verwendeten HTTP Error Codes der Dropbox API aufgelistet. [Dro143]

Code	Description
400	Bad input parameter. Error message should indicate which one and why.
401	Bad or expired token. This can happen if the user or Dropbox revoked or expired an access token. To fix, you should re-authenticate the user.
403	Bad OAuth request (wrong consumer key, bad nonce, expired timestamp...). Unfortunately, re-authenticating the user won't help here.
404	File or folder not found at the specified path.
405	Request method not expected (generally should be GET or POST).
429	Your app is making too many requests and is being rate limited. 429s can trigger on a per-app or per-user basis.
503	If the response includes the Retry-After header, this means your OAuth 1.0 app is being rate limited. Otherwise, this indicates a transient server error, and your app should retry its request.
507	User is over Dropbox storage quota.
5xx	Server error. Check DropboxOps.

Anhang C: Google Drive API Methoden

Nachfolgend sind die wichtigsten Befehle der Google Drive API aufgelistet und im Detail beschrieben. [Goo145]

Alle Befehle werden durch HTTP Methoden ausgeführt.

POST /files/<fileId>/copy
Kopiert einen Ordner oder eine Datei innerhalb des Google Drive. Die Datei wird dabei ohne Zielverzeichnis kopiert. Dieses muss danach der neuen Datei erst zugewiesen werden.

Als verpflichtender Parameter wird die *fileId* des zu kopierenden Objektes übergeben.

DELETE /files/<fileId>
Löscht einen Ordner oder eine Datei innerhalb des Google Drive.

Als verpflichtender Parameter wird die *fileId* des zu löschenden Objektes übergeben.

GET /files/<fileId>
Ruft eine die Metadaten einer Datei ab.

Als verpflichtender Parameter wird die *fileId* der gewünschten Datei übergeben.

Als Rückgabewert wird eine JSON Struktur mit den Metadaten der Datei zurückgeliefert. Die wichtigsten Elemente darin sind:

- createdDate Erstellungsdatum und Uhrzeit der Datei,
- downloadUrl URL zum Download der Datei,
- fileSize Dateigröße in Byte,
- id eindeutige Identifikation der Datei in Google Drive,
- md5Checksum Hashwert der Datei,
- mimeType MIME Typ der Datei,
- modiffiedDate Datum und Uhrzeit der letzte Änderung,
- parents[] Elternobjekte der Datei (Verzeichnisse),
- properties[] Dateieigenschaften und
- title Name der Datei.

POST /files
Fügt eine neue Datei hinzu.

Als verpflichtender Parameter wird der *uploadType* der neuen Datei übergeben. Dieser kann media, multipart oder resumable sein.

Der Inhalt der hochzuladenden Datei befindet sich dabei im RequestBody.

GET /files
Ruft eine Dateiliste des Google Drive ab. Diese Abfrage kann durch ein Such-Query (Parameter *q)* eingeschränkt werden. Ohne Query werden alle Dateien des Google Drive ausgegeben. Die Anzahl der zurückgegebenen Einträge ist durch den Parameter *maxResults* beschränkt. Der Standardwert liegt hier bei 100 Rückgabewerten.

Der Rückgabewert ist eine JSON Struktur der Form:

- kind — hat immer den Wert: drive#fileList,
- etag — ETag der Liste,
- selfLink — Link zur Liste selbst,
- nextPageToken — Wenn die Rückgabe weniger Einträge hat, als die Abfrage ergibt, wird ein Page Token für die nächste Abfrageseite zurückgegeben,
- nextLink — Link zur nächsten Seite der Abfrage und
- items[] — Liste der Metadaten aller abgefragten Dateien (siehe GET /files/<fileId>)

PUT /files/<fileId>
Führt ein Update einer bestehenden Datei durch.

Als verpflichtender Parameter wird die *fileId* der zu aktualisierenden und der *uploadType* der neuen Datei übergeben. Dieser kann media, mulipart oder resumable sein.

Der Inhalt der hochzuladenden Datei befindet sich dabei im Requestbody.

GET /changes/<changeId>
Liefert Informationen zu einer bestimmten Änderung.

Als verpflichtender Parameter wird hier die *changeId* der gewünschten Änderung übergeben.

Der Rückgabewert ist eine JSON Struktur mit folgenden Elementen:

- kind — hat immer den Wert: drive#changeList,
- etag — ETag der Liste,
- selfLink — Link zur Liste selbst,
- fileID — Identifikation der geänderten Datei,
- feleted — gibt an, ob es sich bei der Änderung um eine Löschung handelt,
- file — Metadaten der geänderten Datei (siehe GET /files/<fileId>),
- modificationDate — Datum und Uhrzeit der Änderung.

GET /changes
Gibt eine Liste aller Änderungen zurück. Mit dem Parameter *startChangeId* kann die ID einer Änderung übergeben werden, ab der die Liste abgefragt werden soll. Wird dieser Parameter nicht übergeben, wird eine Liste aller Änderung seit Bestehen des Google Drive zurückgeliefert.

Der Rückgabewert ist eine JSON Struktur mit folgenden Elementen:

- kind hat immer den Wert: drive#change,
- id Identifikation des Änderungseintrages,
- selfLink selfLink Link zur Liste selbst,
- nextPageToken Wenn die Rückgabe weniger Einträge hat, als die Abfrage ergibt, wird ein Page Token für die nächste Abfrageseite zurückgegeben,
- nextLink Link zur nächsten Seite der Abfrage,
- lagestChangeId Letzte Änderungs ID der Liste und
- items[] Liste der Änderungen (siehe GET /changes/<changeId>).

GET /about
Liefert Informationen zum Google Account zurück.

Die Rückgabe erfolgt in Form einer JSON Struktur. Die wichtigsten Elemente sind:

- largestChangeId höchste Change ID des Accounts,
- name Name des Users und
- user[] Userdaten des Users.

Anhang D: Sourcecode des Cloud Storage Pool (CSP)

Nachfolgend ist der Sourcecode des, in dieser Arbeit entwickelten, Prototyps für die Vernetzung von Speicherdiensten in der Cloud angeführt. Der Genaue Aufbau der Anwendung und die Beschreibung der wesentlichen Codeteile befinden sich in Kapitel 4 dieser Arbeit.

Der Sourcecode besteht aus den Teilen

- D.1. Allgemeine Konfiguration: config.inc.php,
- D.2. Allgemeine Funktionen: functions.inc.php,
- D.3. Autorisierung für Dropbox: dropbox_auth.php,
- D.4. Autorisierung für Google Drive: googledrive_auth.php,
- D.5. Webzugriff für Anwender: index.php,
- D.6. Dateidownload für Anwender: getfile.php,
- D.7. Synchronisation der Datenspeicher: syncall.php,
- D.8. WebDAV Server: webdav.php,
- D.9. Klasse Datastorage: classes/Datastorage.php,
- D.10. Klasse Datastorage_Dropbox: classes/Datastorage_Dropbox.php,
- D.11. Klasse Datastorage_Googledrive: classes/Datastorage_Googledrive.php,
- D.12. Klasse CspWebdav: classes/CspWebdav.php.

D.1. Allgemeine Konfiguration: config.inc.php

```
001 <?
002     //Dropbox Parameter
003     $config_dropbox_appinfo = array(
004             "key"       => "DROPBOX_APP_KEY",
005             "secret"    => "DROPBOX APP SECRET"
006         );
007
008     //Google Drive Parameter
009     $config_googledrive_appinfo = array(
010             "client_id" => "GOOGLE_APPID.apps.googleusercontent.com",
011             "client secret" => "GOOGLE APP SECRET"
012     );
013
014     //Appinfos in Array speichern
015     $config_appinfos = array();
016     $config_appinfos["dropbox"] = $config_dropbox_appinfo;
017     $config appinfos["googledrive"] = $config googledrive appinfo;
018
019     //Datenbank Parameter
020     $db = mysql_connect("localhost", "DB_USER", "DB_PWD")
021         or die("Keine DB Verbindung möglich.");
022     mysql_select_db("DB_NAME", $db);
023     /* --------------------------------------------------
024     --
025     -- Tabellenstruktur für Tabelle `csp_datastorages`
026     --
027     CREATE TABLE IF NOT EXISTS `csp_datastorages` (
028       `id` int(11) NOT NULL,
029       `name` varchar(255) NOT NULL,
030       `type` varchar(5) NOT NULL,
031       `ext_root` varchar(255) NOT NULL,
032       `token` varchar(255) NOT NULL,
033       `primary_storage_id` int(11) DEFAULT NULL,
034       `delta value` varchar(255) DEFAULT NULL,
035       `crypto key` varchar(255) DEFAULT NULL,
036       PRIMARY KEY (`id`),
037       UNIQUE KEY `name` (`name`)
038     ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
039
040     --
041     -- Musterdatensatz für Tabelle `csp datastorages`
042     --
043     INSERT INTO `csp_datastorages`
044       (`id`, `name`, `type`, `ext_root`, `token`, `primary_storage_id`, `delta_value`,
`crypto_key`) VALUES
045       (1, 'Dropbox Ordner 1', 'DRPBO', '/Ordner 1', 'ZUGRIFFSTOKEN HIER EINFÜGEN ', NULL,
NULL, NULL)
046
047     --
048     -- Tabellenstruktur für Tabelle `csp_datastorages_types`
049     --
050
051     CREATE TABLE IF NOT EXISTS `csp_datastorages_types` (
052       `id` varchar(5) NOT NULL,
053       `name` varchar(255) NOT NULL,
054       PRIMARY KEY (`name`)
055     ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
056
057     --
058     -- Daten für Tabelle `csp_datastorages_types`
059     --
060
061     INSERT INTO `csp datastorages types` (`id`, `name`) VALUES
062     ('DRPBO', 'Dropbox'),
063     ('GOODR', 'Google Drive');
064
065     */
066
067 ?>
```

D.2. Allgemeine Funktionen: functions.inc.php

```
001 <?php
002     require_once "config.inc.php";
003     require_once "classes/Datastorage.php";
004     require_once "classes/Datastorage_Dropbox.php";
005     require_once "classes/Datastorage_Googledrive.php";
006 
007 
008     /**
009     * getDatastorageByName
010     *
011     * Liefert eine Instanz der Klasse Datastorage anhand des Datastorage Namens zurück.
012     *
013     * @param String     $name               Name des Datastorages
014     * @param Arra       $config appinfos    Appinfos der externen Datenspeicher
(definiert in config.inc.php)
015     */
016     function getDatastorageByName($name,$config_appinfos){
017         $res = mysql_query("select id,name,type,ext_root,token,crypto_key from
csp datastorages
018                             where primary_storage_id is null
019                             and name='".$name."'");
020         if($res && $arr=mysql_fetch_array($res)){
021             switch ($arr["type"]) {
022                 case "DRPBO":
023                     $ds = new
Datastorage_Dropbox($arr["id"],$arr["name"],$arr["ext_root"],$arr["token"],$arr["crypto_key"
],$config_appinfos["dropbox"]);
024                     break;
025                 case "GOODR":
026                     $ds = new
Datastorage_Googledrive($arr["id"],$arr["name"],$arr["ext_root"],$arr["token"],$arr["crypto_
key"],$config appinfos["googledrive"]);
027                     break;
028                 default:
029                     unset($ds);
030                     break;
031             }
032         }
033         if(isset($ds))
034             return $ds;
035         return null;
036     }
037 
038     /**
039     * getDatastorageById
040     *
041     * Liefert eine Instanz der Klasse Datastorage anhand der Datastorage Id zurück.
042     *
043     * @param String     $Id                 ID des Datastorages
044     * @param Arra       $config_appinfos    Appinfos der externen Datenspeicher
(definiert in config.inc.php)
045     */
046     function getDatastorageById($id,$config_appinfos){
047         $res = mysql_query("select id,name,type,ext_root,token,crypto_key from
csp_datastorages
048                     where id=".$id."
049                     ");
050         if($res && $arr=mysql_fetch_array($res)){
051             switch ($arr["type"]) {
052                 case "DRPBO":
053                     $ds = new
Datastorage Dropbox($arr["id"],$arr["name"],$arr["ext root"],$arr["token"],$arr["crypto key"
],$config appinfos["dropbox"]);
054                     break;
055                 case "GOODR":
056                     $ds = new
Datastorage Googledrive($arr["id"],$arr["name"],$arr["ext root"],$arr["token"],$arr["crypto
key"],$config appinfos["googledrive"]);
057                     break;
058                 default:
059                     unset($ds);
```

```
060                     break;
061                 }
062             }
063             if(isset($ds))
064                 return $ds;
065             return null;
066         }
067
068
069 ?>
```

D.3. Autorisierung für Dropbox: dropbox_auth.php

```
001 <html>
002 <head>
003     <meta charset="UTF-8">
004     <title>Cloud Storage Pool (CSP) - Dropbox Autorisierung</title>
005 </head>
006 <body>
007     <h1>Cloud Storage Pool (CSP) - Dropbox Autorisierung</h1>
008 <?php
009     require_once "config.inc.php";
010     require_once "api/Dropbox/autoload.php";
011
012     use \Dropbox as dbx;
013
014     $appInfo = dbx\AppInfo::loadFromJson($config dropbox appinfo);
015     $webAuth = new dbx\WebAuthNoRedirect($appInfo, "PHP-Example/1.0");
016
017     if(isset($_POST['auth_code'])){
018         list($accessToken, $dropboxUserId) = $webAuth->finish($_POST['auth_code']);
019         echo "Autorisierung der App ist erfolgt. <br>";
020         echo "Bitte tragen Sie den unten stehenden Access Token in die CSP
Konfigruration ein. <br>";
021         echo "Access Token: " . $accessToken . "\n";
022     } else {
023         $authorizeUrl = $webAuth->start();
024         if($authorizeUrl){
025             echo "1. Klick auf <a href='".$authorizeUrl."' target='_blank'>Link</a>,
Autorisierung der App und Kopieren des Codes<br />";
026             echo "2. Den Code hier einfügen und absenden:<br /><br />";
027             echo "<form method='post'>";
028             echo "  <input type='text' name='auth_code''>";
029             echo "  <input type='submit' value='Code absenden' class='button' />";
030             echo "</form>";
031             echo "3. Den anschließend angezeigten Access Token in die CSP Konfigruration
einfügen.<br /><br />";
032
033         } else {
034             echo "Keine gültige AuthURL erhelten. Bitte Key und Secret prüfen.";
035         }
036     }
037
038 ?>
039 </body>
040 </html>
```

D.4. Autorisierung für Google Drive: googledrive_auth.php

```
001 <html>
002 <head>
003     <meta charset="UTF-8">
004     <title>Cloud Storage Pool (CSP) - Google Drive Autorisierung</title>
005 </head>
006 <body>
007     <h1>Cloud Storage Pool (CSP) - Google Drive Autorisierung</h1>
008 <?php
009 require_once 'config.inc.php';
010 set_include_path("api/google-api-php-client/src" . PATH_SEPARATOR . get_include_path());
011 require_once 'Google/Client.php';
012
013 $client id = $config googledrive appinfo["client id"];
014 $client secret = $config googledrive appinfo["client secret"];
015 $redirect_uri = 'https://CSPSERVERPFAD/csp/googledrive_auth.php';
016
017 $client = new Google_Client();
018 $client->setClientId($client id);
019 $client->setClientSecret($client secret);
020 $client->setRedirectUri($redirect_uri);
021 $client->setAccessType("offline");
022 $client->addScope("https://www.googleapis.com/auth/drive");
023
024 $authUrl = $client->createAuthUrl();
025
026 if (isset($_GET['code'])) {
027     $client->authenticate($_GET['code']);
028     echo "Autorisierung der App ist erfolgt. <br>";
029     echo "Bitte tragen Sie den unten stehenden Access Token in die CSP Konfigruration
ein. <br>";
030     echo "Access Token: " . $client->getAccessToken() . "\n";
031 } else {
032     if(isset($authUrl)){
033         echo "1. Klick auf <a href='".$authUrl."' target='_blank'>Link</a>,
Autorisierung der App und Kopieren des Codes<br />";
034         echo "2. Den anschließend angezeigten Access Token in die CSP Konfigruration
einfügen.<br /><br />";
035     } else {
036         echo "Keine gültige AuthURL erhalten. Bitte ID und Secret prüfen.";
037     }
038 }
039
040 ?>
041 </body>
042 </html>
```

D.5. Webzugriff für Anwender: index.php

```
001 <html>
002 <head>
003     <meta charset="UTF-8">
004     <title>Cloud Storage Pool (CSP)</title>
005     <script type="text/javascript">
006         function confirmDelete(datei) {
007                 var message = "Wollen Sie die Datei wirklich löschen?";
008                 if(datei!=null) {
009                     message = "Wollen Sie die Datei "+datei+" wirklich löchen?";
010                 }
011 
012                 var del = confirm(message);
013                 return del;
014         }
015     </script>
016 </head>
017 <body>
018     <h1>Cloud Storage Pool (CSP)</h1>
019 <?php
020     require_once "config.inc.php";
021     require_once "functions.inc.php";
022     require_once "classes/Datastorage.php";
023     require_once "classes/Datastorage_Dropbox.php";
024     require_once "classes/Datastorage_Googledrive.php";
025 
026     //Übergabeparameter zuweisen
027     if(isset($_GET['ds_id']))
028         $ds_id = htmlentities($_GET['ds_id']);
029     if(isset($_GET['ds_path']) and $_GET['ds_path']!="/" and $_GET['ds_path']!=null){
030         $ds_path = urldecode($_GET['ds_path']);
031     } else {
032         $ds_path = "";
033     }
034     if(isset($_GET['ds_delete']))
035         $ds_delete = urldecode($_GET['ds_delete']);
036 
037     if(isset($ds_id)){ //Datastorage anzeigen
038         //Parameter des Datastorages laden
039         $ds = getDatastorageById(mysql_real_escape_string($ds_id),$config_appinfos);
040         if($ds){
041             //Dateiupload durchführen
042             if(isset($_FILES["userfile"])){
043                 $message = "";
044                 switch( $_FILES['userfile']['error'] ) {
045                     case UPLOAD_ERR_OK:
046                         $message = false;
047                         break;
048                     case UPLOAD_ERR_INI_SIZE:
049                         $message .= ' - INI_SIZE: file too large (limit of
'.ini_get('upload_max_filesize').').';
050                         break;
051                     case UPLOAD_ERR_FORM_SIZE:
052                         $message .= ' - FORM_SIZE: file too large (limit of
'.ini_get('upload_max_filesize').').';
053                         break;
054                     case UPLOAD_ERR_PARTIAL:
055                         $message .= ' - file upload was not completed.';
056                         break;
057                     case UPLOAD_ERR_NO_FILE:
058                         $message .= ' - zero-length file uploaded.';
059                         break;
060                     default:
061                         $message .= ' - internal error #'.$ FILES['newfile']['error'];
062                         break;
063                 }
064                 echo $message;
065 
066                 $tmpfile = $ FILES["userfile"]["tmp name"];
067 
068                 if ($_FILES["userfile"]["error"] == UPLOAD_ERR_OK) {
```

```
069                     $result = $ds-
>uploadFile($tmpfile,$ds path."/".$ FILES["userfile"]["name"]);
070                     echo "Upload von
".htmlentities(utf8_decode($_FILES["userfile"]["name"])." abgeschlossen.";
071                 }
072             }
073
074             //Ordner erstellen
075             if(isset($_POST["newfolder"])){
076                 $result = $ds->createFolder($ds path."/".$ POST["newfolder"]);
077                 echo "Ordner ".htmlentities(utf8_decode($_POST["newfolder"]))."
angelegt.";
078             }
079
080             //Dateilöschung durchführen
081             if(isset($ds_delete)){
082                 $result = $ds->deleteFile($ds path."/".$ds delete);
083                 echo htmlentities(utf8_decode($ds_delete))." gelöscht.";
084             }
085
086
087             //Ordnerinhalt ausgeben
088
089             //Titelzeile mit Ordnerinfo augeben
090             echo "<h2>/".$ds->getName().(strlen($ds_path)>0?"/".$ds_path:"")."</h2>";
091
092             //Inhaltsliste abrufen
093             $fileList = $ds->getFilelist($ds path);
094
095             //Sortieren der Liste für die Ausgabe (Sortierfunktion)
096             function cmp($a, $b) {
097                 if($a['is_dir'] == $b['is_dir']) {
098                   if($a['name'] != $b['name']){
099                     return strcmp($a['name'], $b['name']);
100                   }
101                   return 0;
102                 }
103                 return ($a['is_dir'] < $b['is_dir']) ? 1 : -1;
104             }
105
106             //Sortieren der Liste für die Ausgabe
107             usort($fileList["contents"], "cmp");
108
109             //Sprungpunkt für übergeordneten Ordner ausgeben
110             if(strlen($ds_path)>0){
111                 $h path = substr($ds path, 0, strlen($ds path)-
strlen(basename($ds path)));
112                 if(strlen($h_path)>0)
113                     $h_path = substr($h_path,0,-1);
114                 echo ".. (<a
href='?ds id=".$ds id."&ds path=".urlencode($h path)."'>Übergeordneten Ordner
öffnen</a>)<br/><br/>\n";
115             } else {
116                 //Wenn bereits im Ordner Root, wieder zurück zur virtuellen
Ordnerauswahl
117                 echo ".. (<a href='?'>Übergeordneten Ordner öffnen</a>)<br/><br/>\n";
118             }
119
120             //Liste Ausgeben
121             foreach ($fileList["contents"] as  $key => $fileListEntry) {
122                 if($fileListEntry["is_dir"]){ //Wenn Ordner
123                     echo $fileListEntry["name"].
124                         " (<a
href='?ds_id=".$ds_id."&ds_path=".urlencode((strlen($ds_path)>0?$ds_path."/":"").$fileListEn
try["name"])."'>Ordner öffnen</a>".
125                         ", <a
href='?ds id=".$ds id."&ds path=".urlencode($ds path)."&ds delete=".urlencode($fileListEntry
["name"])."' onclick=\"return
confirmDelete('".htmlentities(utf8_decode($fileListEntry["name"]))."')\">löschen</a>)<br/>\n
";
126                 } else { //Wenn Datei
127                     echo $fileListEntry["name"].
```

```
128                         " (<a
href='getfile.php?ds_id=".$ds_id."&ds_path=".urlencode((strlen($ds_path)>0?$ds_path."/":"").
$fileListEntry["name"])."'>download</a>".
129                         ", <a
href='?ds_id=".$ds_id."&ds_path=".urlencode($ds_path)."&ds_delete=".urlencode($fileListEntry
["name"])."' onclick=\"return
confirmDelete('".htmlentities(utf8_decode($fileListEntry["name"]))."')\">löschen</a>)<br/>\n
";
130                 }
131             }
132
133             //Formular für Dateiupload ausgeben
134             echo "<br/>";
135             echo "<form enctype='multipart/form-data'
action='?ds_id=".$ds_id."&ds_path=".urlencode($ds_path)."' method='POST'>";
136             echo "    Datei hochladen: <input name='userfile' type='file' />";
137             echo "    <input type='submit' value='Datei hochladen' />";
138             echo "</form>";
139
140             //Formular für neue Unterordner ausgeben
141             echo "<br/>";
142             echo "<form enctype='multipart/form-data'
action='?ds id=".$ds id."&ds path=".urlencode($ds path)."' method='POST'>";
143             echo "    Ordner erstellen: <input name='newfolder' type='text' />";
144             echo "    <input type='submit' value='Ordner erstellen' />";
145             echo "</form>";
146
147         } else { //Datastorage existiert nicht
148             echo "Der Datenspeicher ist nicht vorhanden.";
149         }
150
151
152     } else { //kein Datastorage ausgewählt. Virtuelle Ordnerliste mit allen Datastorages
anzeigen.
153         echo "<h2>Virtuelle Ordnerliste</h2>\n";
154         echo "<h3>Primäre Datenspeicher</h3>\n";
155         //Alle primären Datastorages laden
156         $res = mysql_query("select id,name from csp_datastorages
157                     where primary_storage_id is null
158                     order by name
159                     ");
160         while($arr=mysql_fetch_array($res)){
161             echo "<a href='?ds_id=".$arr["id"]."'>".$arr["name"]."</a><br/>\n";
162         }
163
164         echo "<h3>Sekundäre Datenspeicher</h3>\n";
165         //Alle primären Datastorages laden
166         $res = mysql_query("select id,name from csp_datastorages
167                     where primary_storage_id is not null
168                     order by name
169                     ");
170         while($arr=mysql_fetch_array($res)){
171             echo "<a href='?ds_id=".$arr["id"]."'>".$arr["name"]."</a><br/>\n";
172         }
173     }
174
175 ?>
176 </body>
177 </html>
```

D.6. Dateidownload für Anwender: getfile.php

```
001 <?php
002     require_once "config.inc.php";
003     require_once "api/Dropbox/autoload.php";
004     require_once "classes/Datastorage.php";
005     require_once "classes/Datastorage_Dropbox.php";
006     require_once "classes/Datastorage_Googledrive.php";
007
008     //Übergabeparameter zuweisen
009     if(isset($ GET['ds id']))
010         $ds_id = htmlentities($_GET['ds_id']);
011     if(isset($_GET['ds_path']) and $_GET['ds_path']!="/" and $_GET['ds_path']!=null){
012         $ds path = urldecode($ GET['ds path']);
013     } else {
014         $ds path = "";
015     }
016
017
018     if(isset($ds_id) && isset($ds_path)){
019         //Parameter des Datastorages laden
020         $res = mysql_query("select id,name,type,ext_root,token,crypto_key from
csp datastorages
021                     where id=".mysql_real_escape_string($ds_id)."
022                     ");
023         if($res && $arr=mysql_fetch_array($res)){
024             switch ($arr["type"]) {
025                 case "DRPBO":
026                     $ds = new
Datastorage Dropbox($arr["id"],$arr["name"],$arr["ext root"],$arr["token"],$arr["crypto key"
],$config dropbox appinfo);
027                     break;
028                 case "GOODR":
029                     $ds = new
Datastorage_Googledrive($arr["id"],$arr["name"],$arr["ext_root"],$arr["token"],$arr["crypto_
key"],$config googledrive appinfo);
030                     break;
031                 default:
032                     unset($ds);
033                     break;
034             }
035
036             //Datenspeicher ausgewählt
037             if(isset($ds)){
038                 $filedata = $ds->getFile($ds_path);
039
040                 //Wenn Datei abrufbar
041                 if($filedata != null && $filedata["is_dir"] == 0){
042                     header('Content-Description: File Transfer');
043                     header('Content-Type: '.$filedata["mime_type"]);
044                     header('Content-Disposition: attachment;
filename='.$filedata["name"]);
045                     header('Expires: 0');
046                     header('Cache-Control: must-revalidate');
047                     header('Pragma: public');
048                     header('Content-Length: ' . $filedata["bytes"]);
049
050                     //Dateiinhalt ausgeben
051                     readfile($filedata["tmpfile"]);
052                     unlink($filedata["tmpfile"]);
053                     exit;
054                 } elseif($filedata["is_dir"] == 1){
055                     //Wenn Ordner > Ordnerdaten ausgeben
056                     print_r($filedata);
057                 }
058
059             } else {
060                 echo "Der Datenspeicher Typ ist unbekannt.";
061             }
062         } else {
063             echo "Der Datenspeicher ist nicht vorhanden.";
064         }
065     } else {
```

```
066         echo "Zu wenig Übergabeparameter.";
067     }
068 ?>
```

D.7. Synchronisation der Datenspeicher: syncall.php

```
001 <?php
002     require_once "config.inc.php";
003     require_once "api/Dropbox/autoload.php";
004     require_once "classes/Datastorage.php";
005     require_once "classes/Datastorage_Dropbox.php";
006     require_once "classes/Datastorage_Googledrive.php";
007
008     //Gesamte Ausgabe als Textdatei
009     header('Content-Description: CSP Sync Log');
010     header('Content-Type: text/plain; charset=UTF-8');
011     header('Expires: 0');
012     header('Cache-Control: must-revalidate');
013     header('Pragma: public');
014
015     //set time limit ( 30 ); //Ausführungszeit bei Bedarf erhöhen
016    echo "*** Synchronisation gestartet. ***\n";
017
018    //Primäre Storages der ersten Ebene abrufen (cd1.primary storage id is null)
019    $res_pri_top = mysql_query("select distinct
cd1.id,cd1.name,cd1.ext_root,cd1.token,cd1.crypto_key,cd1.type,cd1.delta_value from
csp_datastorages cd1
020                        join csp_datastorages cd2 on(cd1.id = cd2.primary_storage_id)
021                        where cd1.primary storage id is null");
022    while($arr_pri_top = mysql_fetch_array($res_pri_top) ){
023         //Variable zurücksetzen, da neues primäres Storage oberster Ebene geladen wird
024         unset($delta done);
025         unset($ds_cursor);
026         echo "\n### Hauptstorage: (".$arr_pri_top["id"].") ".$arr_pri_top["name"]."
###\n";
027         while($res_pri = mysql_query("select distinct
cd1.id,cd1.name,cd1.ext_root,cd1.token,cd1.crypto_key,cd1.type,cd1.delta_value from
csp_datastorages cd1
028                           join csp_datastorages cd2 on(cd1.id = cd2.primary_storage_id)
029                           where cd1.primary storage id ".
030                           (!isset($ds_cursor)?"is null and cd1.id=".$arr_pri_top["id"]:"in
(".$ds cursor.")"))
031          ){
032             $ds_cursor = "";
033             while($res_pri && $arr_pri=mysql_fetch_array($res_pri)){
034                 echo "> (".$arr_pri["id"].") ".$arr_pri["name"];
035
036                 if(strlen($ds_cursor)==0)
037                     $ds_cursor = $arr_pri["id"];
038                 else
039                     $ds_cursor .= ",".$arr_pri["id"];
040
041                 //Primäres Datastorage instanzieren
042                 switch ($arr_pri["type"]) {
043                     case "DRPBO":
044                         $ds_pri = new
Datastorage Dropbox($arr pri["id"],$arr pri["name"],$arr pri["ext root"],$arr pri["token"],$
arr pri["crypto key"],$config dropbox appinfo);
045                         break;
046                     case "GOODR":
047                         $ds_pri = new
Datastorage_Googledrive($arr_pri["id"],$arr_pri["name"],$arr_pri["ext_root"],$arr_pri["token
"],$arr pri["crypto key"],$config googledrive appinfo);
048                         break;
049                     default:
050                         unset($ds_pri);
051                         break;
052                 }
053                 if(isset($ds_pri)){
054
055                     //Deltaliste des primären Datastorages abrufen
056                     $delta_pri = $ds_pri->getDelta($arr_pri["delta_value"]);
057
058                     //print r($delta pri); //Anzeige der Deltawerte bei Bedarf
aktivieren
059
060                     //Zugehöriges sekundäres Datastorage laden
```

```
061                     $res_sec = mysql_query("select distinct
id,name,ext_root,token,crypto_key,type,delta_value from csp_datastorages
062                             where primary_storage_id = ".$arr_pri["id"].";
063                             ");
064                     if($res_sec && $arr_sec=mysql_fetch_array($res_sec)){
065                         echo " <<>> (".$arr_sec["id"].") ".$arr_sec["name"]."\n";
066
067                         //Sekundäres Datastorage instanzieren
068                         switch ($arr_sec["type"]) {
069                             case "DRPBO":
070                                 $ds_sec = new
Datastorage Dropbox($arr sec["id"],$arr sec["name"],$arr sec["ext root"],$arr sec["token"],$
arr sec["crypto key"],$config dropbox appinfo);
071                                 break;
072                             case "GOODR":
073                                 $ds_sec = new
Datastorage Googledrive($arr sec["id"],$arr sec["name"],$arr sec["ext root"],$arr sec["token
"],$arr sec["crypto key"],$config googledrive appinfo);
074                                 break;
075                             default:
076                                 unset($ds_sec);
077                                 break;
078                         }
079                         if(isset($ds_sec)){
080                             //Deltaliste des sekundären Datastorages abrufen
081                             $delta_sec = $ds_sec->getDelta($arr_sec["delta_value"]);
082
083                             //print r($delta sec); //Anzeige der Deltawerte bei Bedarf
aktivieren
084                         } else {
085                             echo "Der Typ des sekundären Datenspeichers
(".$arr_pri["id"].") ".$arr_pri["name"]." ist nicht vorhanden.\n";
086                         }
087                     }
088                 } else {
089                     echo "Der Typ des primären Datenspeichers (".$arr_pri["id"].")
".$arr_pri["name"]." ist nicht vorhanden.\n";
090                 }
091
092                 //Wenn bereits eine Synchronisation von einem höherwertigen Datastorage
erfolgt ist,
093                 // muss diese nach unten weitergegeben werden
094                 if(isset($delta done)){
095                     foreach ($delta_done as $key => $value) {
096                         $delta pri["entries"][] = $value;
097                     }
098                 }
099
100                 //Alle abgearbeiteten Deltas um Doppelbearbeitungen zu verhindern
101                 $delta_done = Array();
102
103                 while( ( isset($delta_pri["entries"]) && count($delta_pri["entries"])>0
)
104                     || ( isset($delta_sec["entries"]) && count($delta_sec["entries"])>0
)){
105                     //Dateikonflikte zwischen primärer und sekundärer Deltaliste suchen
106                     if(isset($delta_pri["entries"]) && isset($delta_sec["entries"])){
107                         foreach ($delta_pri["entries"] as $key_pri => $value_pri) {
108                             foreach ($delta_sec["entries"] as $key_sec => $value_sec) {
109                                 //Wenn beide Dateien geändert wurden
110                                 if( $value_pri["path"]==$value_sec["path"] &&
111                                     $value sec["action"]=="upload"){
112                                     //Konflikt erkannt (Datei wurde auf primärem
Datastorage geändert oder gelöscht)
113                                     //Datei von sekundärem Datastorage als Konflikdatei
in primären Datastorage kopieren
114                                     try {
115                                         $filedata = $ds sec-
>getFile("/".$value sec["path"]);
116                                         $konflict_path =
"/KONFLIKT".date('YmdHis')."_".$value_pri["path"];
117                                         $ds pri-
>uploadFile($filedata["tmpfile"],$konflict_path);
```

```
118                                         $ds_sec-
>uploadFile($filedata["tmpfile"],$konflict_path);
119                                         unlink($filedata["tmpfile"]);
120
121                                         $delta_done_entry["action"] = "upload";
122                                         $delta_done_entry["path"] = $konflict_path;
123                                         $delta_done[] = $delta_done_entry;
124
125                                         echo "Konfliktdatei '".$value_sec["path"]."' von
'".$arr_pri["id"].", ".$arr_pri["name"]."' hochgeladen.\n";;
126                                     } catch (Exception $e) {
127                                         echo "Konfliktdatei '".$value_sec["path"]."' von
'".$arr_pri["id"].", ".$arr_pri["name"]."' kann nicht hochgeladen werden.\n";
128                                     }
129                                     //Änderungseintrag aus sekundärer Deltaliste
entfernen
130                                     unset($delta_sec["entries"][$key_sec]);
131                                 } elseif($value_pri["path"]==$value_sec["path"] &&
132                                     $value_pri["action"]=="upload" &&
133                                     $value_sec["action"]=="delete"){
134                                     //Datei wurde auf sekundärem Datastorade gelöscht
135                                     //Kann ignoriert werden, da sie auf primärem
Datastorage geändert wurde
136                                     //Änderungseintrag aus sekundärer Deltaliste
entfernen
137                                     unset($delta_sec["entries"][$key_sec]);
138                                 }
139                             }
140                         }
141                     }
142
143                     //Deltas der primären Liste in sekundärem Storage ausführen
144                     if(isset($delta_pri["entries"])){
145                         foreach ($delta_pri["entries"] as $key => $value) {
146                             switch ($value["action"]) {
147                                 case "delete":
148                                     try {
149                                         $ds_sec->deleteFile("/".$value["path"]);
150                                         echo "Datei/Ordner '".$value["path"]."' von
'".$arr_sec["id"].", ".$arr_sec["name"]."' geloescht.\n";
151                                     } catch (Exception $e) {
152                                         echo "Datei/Ordner '".$value["path"]."' von
'".$arr_sec["id"].", ".$arr_sec["name"]."' kann nicht geloescht werden.\n";
153                                     }
154                                     break;
155                                 case "newdir":
156                                     try {
157                                         $ds_sec->createFolder("/".$value["path"]);
158                                         echo "Ordner '".$value["path"]."' von
'".$arr_sec["id"].", ".$arr_sec["name"]."' angelegt.\n";
159                                     } catch (Exception $e) {
160                                         echo "Ordner '".$value["path"]."' von
'".$arr_sec["id"].", ".$arr_sec["name"]."' kann nicht angelegt werden.\n";
161                                     }
162                                     break;
163                                 case "upload":
164                                     try {
165                                         $filedata = $ds_pri-
>getFile("/".$value["path"]);
166
167                                         if(!$filedata && !isset($filedata["tmpfile"])){
168                                             throw new Exception("Datei nicht vorhanden",
1);
169                                         }
170                                         $ds_sec-
>uploadFile($filedata["tmpfile"],"/".$value["path"]);
171                                         unlink($filedata["tmpfile"]);
172
173                                         echo "Datei '".$value["path"]."' auf
'".$arr_sec["id"].", ".$arr_sec["name"]."' hochgeladen.\n";;
174                                     } catch (Exception $e) {
175                                         echo "Datei '".$value["path"]."' auf
'".$arr_sec["id"].", ".$arr_sec["name"]."' kann nicht hochgeladen werden.\n";
176                                     }
```

```
177                                     break;
178                                 default:
179
180                                     break;
181                             }
182                             $delta_done[] = $value;
183                         }
184                     }
185
186                     //Deltas der sekundären Liste in primären Storage ausführen
187                     if(isset($delta_sec["entries"])){
188                         foreach ($delta_sec["entries"] as $key => $value) {
189                             switch ($value["action"]) {
190                                 case "delete":
191                                     try {
192                                         $ds_pri->deleteFile("/".$value["path"]);
193                                         echo "Datei/Ordner '".$value["path"]."' von
'".$arr_pri["id"].", ".$arr_pri["name"]."' geloescht.\n";
194                                     } catch (Exception $e) {
195                                         echo "Datei/Ordner '".$value["path"]."' von
'".$arr_pri["id"].", ".$arr_pri["name"]."' kann nicht geloescht werden.\n";
196                                     }
197                                     break;
198                                 case "newdir":
199                                     try {
200                                         $ds_pri->createFolder("/".$value["path"]);
201                                         echo "Ordner '".$value["path"]."' von
'".$arr_pri["id"].", ".$arr_pri["name"]."' angelegt.\n";
202                                     } catch (Exception $e) {
203                                         echo "Ordner '".$value["path"]."' von
'".$arr_pri["id"].", ".$arr_pri["name"]."' kann nicht angelegt werden.\n";
204                                     }
205                                     break;
206                                 case "upload":
207                                     try {
208                                         $filedata = $ds_sec-
>getFile("/".$value["path"]);
209
210                                         if(!$filedata && !isset($filedata["tmpfile"])){
211                                             throw new Exception("Datei nicht vorhanden",
1);
212                                         }
213                                         $ds_pri-
>uploadFile($filedata["tmpfile"],"/".$value["path"]);
214                                         unlink($filedata["tmpfile"]);
215
216                                         echo "Datei '".$value["path"]."' auf
'".$arr_pri["id"].", ".$arr_pri["name"]."' hochgeladen.\n";;
217                                     } catch (Exception $e) {
218                                         echo "Datei '".$value["path"]."' auf
'".$arr_pri["id"].", ".$arr_pri["name"]."' kann nicht hochgeladen werden.\n";
219                                     }
220                                     break;
221                                 default:
222
223                                     break;
224                             }
225                             $delta_done[] = $value;
226                         }
227                     }
228
229                     //Deltaliste des sekundären Storages neu laden und speichern
230                     $delta_sec = $ds_sec->getDelta($delta_sec["cursor"]);
231                     if(mysql_query("update csp_datastorages
232                                         set delta_value='".$delta_sec["cursor"]."'
233                                         where id = ".$arr_sec["id"].";
234                                         ")){
235                         echo "Neuer Delta Cursor für (".$arr_sec["id"].")
".$arr_sec["name"]." gespeichert.\n";
236                     }else{
237                         echo "Neuer Delta Cursor für (".$arr_sec["id"].")
".$arr_sec["name"]." kann nicht gespeichert werden.\n";
238                     }
239
```

```
240
241                     //Deltaliste des primären Storages neu laden und speichern
242                     $delta_pri = $ds_pri->getDelta($delta_pri["cursor"]);
243
244                     if(mysql_query("update csp_datastorages
245                                         set delta value='".$delta pri["cursor"]."'
246                                         where id = ".$arr pri["id"].";
247                                     ")){
248                         echo "Neuer Delta Cursor für (".$arr_pri["id"].")
".$arr pri["name"]." gespeichert.\n";
249                     }else{
250                         echo "Neuer Delta Cursor für (".$arr_pri["id"].")
".$arr pri["name"]." kann nicht gespeichert werden.\n";
251                     }
252
253
254                     //Deltalisten um die bereits abgearbeiteten Deltas bereinigen
255                     if(isset($delta_pri["entries"])){
256                         foreach($delta_pri["entries"] as $key => $value) {
257                             foreach($delta done as $key done => $value_done)
258                             if($value["path"]==$value_done["path"] &&
$value["action"]==$value done["action"]){
259                                 unset($delta_pri["entries"][$key]);
260                             }
261                         }
262                         if(count($delta_pri["entries"])==0)
263                             unset($delta_pri["entries"]);
264                     }
265                     if(isset($delta_sec["entries"])){
266                         foreach($delta_sec["entries"] as $key => $value) {
267                             foreach($delta done as $key done => $value_done)
268                             if($value["path"]==$value_done["path"] &&
$value["action"]==$value done["action"]){
269                                 unset($delta_sec["entries"][$key]);
270                             }
271                         }
272                         if(count($delta sec["entries"])==0)
273                             unset($delta_sec["entries"]);
274                     }
275
276                 }
277
278             }
279         }
280     }
281     echo "\n*** Synchronisation beendet. ***";
282 ?>
```

D.8. WebDAV Server: webdav.php

```
001 <?php
002
003     require_once "config.inc.php";
004     require_once "classes/CspWebdav.php";
005
006     $server = new CspWebdav();
007     set time limit ( 60*2 ); //Ausführungszeit auf 2 Minuten erhöhen
008     $server->setConfigAppinfos($config appinfos);
009     $server->ServeRequest();
010
011 ?>
```

D.9. Klasse Datastorage: classes/Datastorage.php

```
001 <?php
002 /**
003  * Datastorage
004  *
005  * Abstrakte Klasse für die einheitliche Insanzierung von Datenspeichern
unterschiedlicher Quelle
006  * als virtueller Datenspeicher des Cloud Storage Pools (CSP)
007  *
008  * @author      Peter Völkl <peter@vape.net>
009  * @package     CSP
010  * @version     2014-04-26
011  */
012 abstract class Datastorage{
013     protected $id;
014     protected $name;
015     protected $type;
016     protected $ext_root;
017     protected $token;
018     protected $crypto_key;
019
020     /**
021     * __construct
022     *
023     * Initialisierung der allgemenen Klassenvariablen
024     *
025     * @param string     $id         Eindeutige ID des virtuellen Datenspeichers
026     * @param string     $name       Eindeutiger Bezeichnung des virtuellen
Datenspeichers
027     * @param string     $ext root   Root Ordner der Instanz innerhalb des externen
Datenspeichers
028     * @param string     $token      Authorisierungs Token zur Verbindung mit dem
externen Datnespeicher
029     * @param string     $crypto_key Passphrase zur Verschlüsselung der Daten innderhalb
des externen Datenspeichers
030     */
031     function __construct($id,$name,$ext_root,$token,$crypto_key){
032         $this->id = $id;
033         $this->name = $name;
034         $this->ext_root = $ext_root;
035         $this->token = $token;
036         $this->crypto key = $crypto key;
037     }
038
039     /**
040     * getName
041     *
042     * Liefert den Namen des Datastorages zurück
043     *
044     * @return string    Name des Datastorages
045     */
046     function getName(){
047         return $this->name;
048     }
049
050     /**
051     * csp_encryptStream
052     *
053     * Verschlüsselt einen Filestream
054     * Nach dem Beispiel von http://www.php.net/manual/en/filters.encryption.php
055     *
056     * @param resource   &$fp        Filepointer des zu verschlüsselnden Filestreams
057     * @param string     $crypto key Verwendeter Schlussel (Passphrase)
058     */
059     function csp_encryptStream(&$fp, $crypto_key){
060         $iv = substr(md5('iv#'.$crypto key, true), 0, 8);
061         $key = substr(md5('pass1#'.$crypto_key, true) .
062                     md5('pass2#'.$crypto_key, true), 0, 24);
063         $opts = array('iv'=>$iv, 'key'=>$key);
064         stream filter append($fp, 'mcrypt.tripledes', STREAM FILTER READ, $opts);
065     }
066
```

```
067
068     /**
069     * csp_decryptStream
070     *
071     * Entschlüsselt einen Filestream
072     * Nach dem Beispiel von http://www.php.net/manual/en/filters.encryption.php
073     *
074     * @param resource   &$fp        Filepointer des zu entschlüsselnden Filestreams
075     * @param string     $crypto_key Verwendeter Schlussel (Passphrase)
076     */
077     function csp_decryptStream(&$fp, $crypto_key){
078         $iv = substr(md5('iv#'.$crypto_key, true), 0, 8);
079         $key = substr(md5('pass1#'.$crypto_key, true) .
080                       md5('pass2#'.$crypto key, true), 0, 24);
081         $opts = array('iv'=>$iv, 'key'=>$key);
082         stream filter append($fp, 'mdecrypt.tripledes', STREAM FILTER WRITE, $opts);
083     }
084
085     /**
086     * getFilelist
087     *
088     * Abfrage aller Dateien und ihrer Eigenschaften eines Verzeichnisses innerhalb des
virtuellen Datenspeichers
089     *
090     * @param string     $directory      Verzeichnispfad
091     *
092     * @return Array     Daten des Ordners oder der Datei $directory inklusive aller
enthaltenen Dateien und Ordner
093     *                   Struktur:
094     *                   $ret["name"]= Name von $directory
095     *                   $ret["path"] = Pfad innerhalb des virtuellen Datenspeichers zu
$directory
096     *                   $ret["datastorage name"] = Eindeutiger Bezeichnung des
virtuellen Datenspeichers
097     *                   $ret["creation date"] = Erstellungsdatum von $directory;
098     *                   $ret["last_modified"] = Datum der letzten Änderung von
$directory;
099     *                   $ret["is_dir"] = [1...wenn es sich um ein Verzeichnis handelt]
100     *                   $ret["mime type"] = MIME Typ von $directory
101     *                   $ret["bytes"] = Größe von $directory in Byte
102     *                   $ret["contents"][] = Dateien und Ordner innrehalb von $directory
103     *                   $ret["contents"][]["name"]= Name des Elements
104     *                   $ret["contents"][]["path"] = Pfad innerhalb des virtuellen
Datenspeichers zum Element
105     *                   $ret["contents"][]["datastorage name"] = Eindeutiger Bezeichnung
des virtuellen Datenspeichers
106     *                   $ret["contents"][]["creation date"] = Erstellungsdatum des
Elements;
107     *                   $ret["contents"][]["last_modified"] = Datum der letzten Änderung
des Elements;
108     *                   $ret["contents"][]["is dir"] = [1...wenn es sich um ein
Verzeichnis handelt]
109     *                   $ret["contents"][]["mime_type"] = MIME Typ des Elements
110     *                   $ret["contents"][]["bytes"] = Größe des Elements in Byte
111     */
112     abstract function getFilelist($directory);
113
114     /**
115     * getFile
116     *
117     * Liefert eine Datei und ihre Eigenschaften zurück
118     *
119     * @param string     $file       Pfad innerhalb des virtuellen Datenspeichers zur
Datei
120     *
121     * @return Array     Eigenschaften der Datei und Pfad zu temporärer Datei für ihren
Abruf
122     *                   Struktur:
123     *                   $ret["name"]= Name von $file
124     *                   $ret["datastorage_name"] = Eindeutiger Bezeichnung des
virtuellen Datenspeichers
125     *                   $ret["creation date"] = Erstellungsdatum von $file;
126     *                   $ret["last modified"] = Datum der letzten Änderung von $file;
127     *                   $ret["is_dir"] = [1...wenn es sich um ein Verzeichnis handelt]
```

```
128     *                   $ret["mime type"] = MIME Typ von $directory
129     *                   $ret["bytes"] = Größe von $directory in Byte
130     *                   $ret["tmpfile"] = Pfad zur angelegten temporärern Datei für
weitere Verarbeitung
131     */
132     abstract function getFile($file);
133
134     /**
135     * uploadFile
136     *
137     * Ladet eine Datei in den virtuellen Datenspeicher hoch
138     *
139     * @param string     $tmpfile        Pfad zur temporären Datei die hochgeladen werden
soll
140     * @param string     $newfile        Pfad und Name der neuen Datei innerhalb des
virtuellen Datenspeichers
141     */
142     abstract function uploadFile($tmpfile,$newfile);
143
144     /**
145     * uploadFileStream
146     *
147     * Lädt eine Datei in den virtuellen Datenspeicher hoch
148     *
149     * @param string     $fp             Filestream der hochzuladenden Datei
150     * @param string     $newfile        Pfad und Name der neuen Datei innerhalb des
virtuellen Datenspeichers
151     */
152     abstract function uploadFileStream($fp,$newfile);
153
154     /**
155     * deleteFile
156     *
157     * Löscht eine Datei des virtuellen Datenspeichers
158     *
159     * @param string     $file       Pfad innerhalb des virtuellen Datenspeichers zur
Datei
160     */
161     abstract function deleteFile($file);
162
163     /**
164     * copyFile
165     *
166     * Kopiert eine Datei innerhalb des Datenspeichers
167     *
168     * @param String     $source file    Pfad der Quelldatei innerhalb des virtuellen
Datenspeichers
169     * @param String     $dest_file  Pfad der Quelldatei innerhalb des virtuellen
Datenspeichers
170     */
171     abstract function copyFile($source_file, $dest_file);
172
173     /**
174     * createFolder
175     *
176     * Erstellt einen neuen Ordner innerhalb des virtuellen Datenspeichers
177     *
178     * @param string     $folder     Pfad des neuen Ordners inkusive Ordnername innerhalb
des virtuellen Datenspeichers
179     */
180     abstract function createFolder($folder);
181
182     /**
183     * getDelta
184     *
185     * Liefert alle Änderungen des virtuellen Datenspeichers seit dem übergebenen Cursor.
186     * Wird null als Cusor übergeben werden alle Änderungen seit bestehen des virtuellen
Datenspeichers zurückgeleifert.
187     *
188     * @param string     $delta_value        Cursor ab dem das Delta berechnet werden
soll
189     *
190     * @return Array     Alle Änderungen des Datenspeichers ab dem übergebenen Cursor
191     *                   Struktur:
```

```
192     *                   $ret["cursor"]= Name von $file
193     *                   $ret["entries"][] = Eindeutiger Bezeichnung des virtuellen
Datenspeichers
194     *                   $ret["entries"][]["path"] = Pfad zur/zum betroffenen
Datei/Ordner
195     *                   $ret["entries"][]["action"] = Durchgeführte Aktion
[delete|newdir|upload]
196     */
197     abstract function getDelta($delta_value);
198
199 }
200
201 ?>
```

D.10. Klasse Datastorage_Dropbox: classes/Datastorage_Dropbox.php

```
001 <?php
002
003 require_once "classes/Datastorage.php";
004 require_once "api/Dropbox/autoload.php";
005
006 /**
007  * Datastorage Dropbox
008  *
009  * Klasse für die Instanzierung von Dropbox-Speichern als virtuelle Datenspeicher des
Cloud Storage Pools (CSP)
010  *
011  * @author      Peter Völkl <peter@vape.net>
012  * @package     CSP
013  * @version     2014-04-26
014  */
015 class Datastorage_Dropbox extends Datastorage{
016     private $appInfo;
017     private $webAuth;
018     private $dbxClient;
019
020     /**
021     * __construct
022     *
023     * Initialisierung der allgemenen Klassenvariablen
024     *
025     * @param string     $id         Eindeutige ID des virtuellen Datenspeichers
026     * @param string     $name       Eindeutiger Bezeichnung des virtuellen
Datenspeichers
027     * @param string     $ext root   Root Ordner der Instanz innerhalb des externen
Datenspeichers
028     * @param string     $token      Authorisierungs Token zur Verbindung mit dem
externen Datnespeicher
029     * @param string     $crypto_key Passphrase zur Verschlüsselung der Daten innderhalb
des externen Datenspeichers
030     * @param Array      $appinfo    Dropbox Appinfo Array("key","secret")
031     */
032     function __construct($id,$name,$ext_root,$token,$crypto_key,$appinfo){
033         parent::__construct($id,$name,$ext_root,$token,$crypto_key);
034         $this->type = "DRPBO";
035
036         //Dropboxzugriff initialisieren
037         $this->appInfo = \Dropbox\AppInfo::loadFromJson($appinfo);
038         $this->webAuth = new \Dropbox\WebAuthNoRedirect($this->appInfo, "PHP-
Example/1.0");
039         $this->dbxClient = new \Dropbox\Client($this->token, "PHP-Example/1.0");
040     }
041
042     /**
043     * getFilelist
044     *
045     * Abfrage aller Dateien und ihrer Eigenschaften eines Verzeichnisses innerhalb des
virtuellen Datenspeichers
046     *
047     * @param string     $directory      Verzeichnispfad
048     *
049     * @return Array     Daten des Ordners oder der Datei $directory inklusive aller
enthaltenen Dateien und Ordner
050     *                   Struktur:
051     *                   $ret["name"]= Name von $directory
052     *                   $ret["path"] = Pfad innerhalb des virtuellen Datenspeichers zu
$directory
053     *                   $ret["datastorage name"] = Eindeutiger Bezeichnung des
virtuellen Datenspeichers
054     *                   $ret["creation date"] = Erstellungsdatum von $directory;
055     *                   $ret["last_modified"] = Datum der letzten Änderung von
$directory;
056     *                   $ret["is dir"] = [1...wenn es sich um ein Verzeichnis handelt]
057     *                   $ret["mime type"] = MIME Typ von $directory
058     *                   $ret["bytes"] = Größe von $directory in Byte
059     *                   $ret["contents"][] = Dateien und Ordner innrehalb von $directory
060     *                   $ret["contents"][]["name"]= Name des Elements
```

```
061     *                   $ret["contents"][]["path"] = Pfad innerhalb des virtuellen
Datenspeichers zum Element
062     *                   $ret["contents"][]["datastorage_name"] = Eindeutiger Bezeichnung
des virtuellen Datenspeichers
063     *                   $ret["contents"][]["creation_date"] = Erstellungsdatum des
Elements;
064     *                   $ret["contents"][]["last modified"] = Datum der letzten Änderung
des Elements;
065     *                   $ret["contents"][]["is_dir"] = [1...wenn es sich um ein
Verzeichnis handelt]
066     *                   $ret["contents"][]["mime type"] = MIME Typ des Elements
067     *                   $ret["contents"][]["bytes"] = Größe des Elements in Byte
068     */
069     function getFilelist($directory){
070         $ret = array();
071
072
073         if(strlen($directory)>0)
074             $path=$this->ext_root."/".$directory;
075         else
076             $path=$this->ext_root;
077
078         $folderMetadata = $this->dbxClient->getMetadataWithChildren($path);
079         $ret["contents"] = Array();
080         if($folderMetadata){
081             $ret["name"]= basename($directory);
082             $ret["path"] = substr($directory, 0, strlen($directory)-
strlen(basename($directory)));
083             if(substr($ret["path"],-1)=="/")
084                 $ret["path"] = substr($ret["path"],0,-1);
085             $ret["datastorage_name"]=$this->name;
086             $ret["creation_date"] = strtotime($folderMetadata["modified"]);
087             $ret["last modified"] = strtotime($folderMetadata["modified"]);
088             $ret["is dir"] = $folderMetadata["is dir"];
089             if(isset($folderMetadata["contents"])){
090                 foreach ($folderMetadata["contents"] as $key => $value) {
091                     $ret["contents"][$key]["name"] =
substr($value["path"],strlen($path)+1);
092                     $ret["contents"][$key]["path"] = $directory;
093                     $ret["contents"][$key]["datastorage name"]=$this->name;
094                     $ret["contents"][$key]["is dir"] = $value["is dir"];
095                     $ret["contents"][$key]["creation_date"] =
strtotime($value["modified"]);
096                     $ret["contents"][$key]["last_modified"] =
strtotime($value["modified"]);
097                     if(isset($value["mime_type"]))
098                         $ret["contents"][$key]["mime type"] = $value["mime type"];
099                     if(isset($value["bytes"]))
100                         $ret["contents"][$key]["bytes"] = $value["bytes"];
101                 }
102             }
103         }
104
105         return $ret;
106     }
107
108     /**
109     * getFile
110     *
111     * Liefert eine Datei und ihre Eigenschaften zurück
112     *
113     * @param string     $file       Pfad innerhalb des virtuellen Datenspeichers zur
Datei
114     *
115     * @return Array     Eigenschaften der Datei und Pfad zu temporärer Datei für ihren
Abruf
116     *                   Struktur:
117     *                   $ret["name"]= Name von $file
118     *                   $ret["datastorage name"] = Eindeutiger Bezeichnung des
virtuellen Datenspeichers
119     *                   $ret["creation_date"] = Erstellungsdatum von $file;
120     *                   $ret["last modified"] = Datum der letzten Änderung von $file;
121     *                   $ret["is dir"] = [1...wenn es sich um ein Verzeichnis handelt]
122     *                   $ret["mime_type"] = MIME Typ von $directory
```

```
123     *                   $ret["bytes"] = Größe von $directory in Byte
124     *                   $ret["tmpfile"] = Pfad zur angelegten temporärern Datei für
weitere Verarbeitung
125     */
126     function getFile($file){
127
128         $local temp = "tmp/";
129
130         if($file=="")
131             $path=$this->ext root;
132         else
133             $path=$this->ext root."/".$file;
134         $filename=basename($path);
135
136         $fp = fopen($local_temp.$filename, "w+b");
137
138         //Entschlüsselung
139         if($this->crypto_key != null){
140             $this->csp_decryptStream($fp, $this->crypto_key);
141         }
142
143         $fileMetadata = $this->dbxClient->getFile($path, $fp);
144         fclose($fp);
145
146         $fileList=null;
147         if(isset($fileMetadata) && $fileMetadata != null){
148             //Ist Datei
149             $ret["name"]=$filename;
150             $ret["datastorage name"]=$this->name;
151             $ret["tmpfile"]=$local_temp.$filename;
152             $ret["mime_type"]=$fileMetadata["mime_type"];
153             $ret["bytes"]=$fileMetadata["bytes"];
154             $ret["creation date"] = strtotime($fileMetadata["modified"]);
155             $ret["last modified"] = strtotime($fileMetadata["modified"]);
156             $ret["is dir"] = 0;
157         } elseif(($fileList = $this->getFilelist($file)) && $fileList["is_dir"]==1){
158             //Ist Verzeichnis
159             $ret["name"]=$filename;
160             $ret["datastorage name"]=$this->ext root;
161             $ret["mime_type"]=null;
162             $ret["bytes"]=0;
163             $ret["creation_date"] = strtotime($fileMetadata["modified"]);
164             $ret["last modified"] = strtotime($fileMetadata["modified"]);
165             if(isset($fileList["contents"]))
166                 $ret["file list"] = $fileList["contents"];
167             $ret["is dir"] = 1;
168         } else {
169             $ret = null;
170         }
171
172         return $ret;
173     }
174
175     /**
176     * uploadFile
177     *
178     * Ladet eine Datei in den virtuellen Datenspeicher hoch
179     *
180     * @param string     $tmpfile         Pfad zur temporären Datei die hochgeladen werden
soll
181     * @param string     $newfile         Pfad und Name der neuen Datei innerhalb des
virtuellen Datenspeichers
182     */
183     function uploadFile($tmpfile,$newfile){
184         $path=$this->ext_root."/".$newfile;
185
186         $fp = fopen($tmpfile, "rb");
187
188         //Verschlüsselung
189         if($this->crypto_key != null){
190             $this->csp_encryptStream($fp, $this->crypto_key);
191         }
192
```

```
193         $result = $this->dbxClient->uploadFile($path, \Dropbox\WriteMode::update(null),
$fp);
194         fclose($fp);
195
196         return $result;
197     }
198
199     /**
200     * uploadFileStream
201     *
202     * Ladet eine Datei in den virtuellen Datenspeicher hoch
203     *
204     * @param string     $fp             Filestream der hochzuladenden Datei
205     * @param string     $newfile        Pfad und Name der neuen Datei innerhalb des
virtuellen Datenspeichers
206     */
207     function uploadFileStream($fp,$newfile){
208         $path=$this->ext root."/".$newfile;
209
210         //Verschlüsselung
211         if($this->crypto_key != null){
212             $this->csp encryptStream($fp, $this->crypto key);
213         }
214
215         $result = $this->dbxClient->uploadFile($path, \Dropbox\WriteMode::update(null),
$fp);
216         fclose($fp);
217
218         return $result;
219     }
220
221     /**
222     * deleteFile
223     *
224     * Löscht eine Datei des virtuellen Datenspeichers
225     *
226     * @param string     $file       Pfad innerhalb des virtuellen Datenspeichers zur
Datei
227     */
228     function deleteFile($file){
229         $path=$this->ext root."/".$file;
230
231         $result = $this->dbxClient->delete($path);
232
233         return $result;
234     }
235
236     /**
237     * copyFile
238     *
239     * Kopiert eine Datei innerhalb des Datenspeichers
240     *
241     * @param String     $source_file    Pfad der Quelldatei innerhalb des virtuellen
Datenspeichers
242     * @param String     $dest_file  Pfad der Quelldatei innerhalb des virtuellen
Datenspeichers
243     */
244     function copyFile($source_file, $dest_file){
245         $source_path=$this->ext_root."/".$source_file;
246         $dest_path=$this->ext_root."/".$dest_file;
247
248         $result = $this->dbxClient->copy($source path,$dest path);
249
250         return $result;
251     }
252
253     /**
254     * createFolder
255     *
256     * Erstellt einen neuen Ordner innerhalb des virtuellen Datenspeichers
257     *
258     * @param string     $folder     Pfad des neuen Ordners inkusive Ordnername innerhalb
des virtuellen Datenspeichers
259     */
```

```
260     function createFolder($folder){
261         $path=$this->ext_root."/".$folder;
262 
263         $result = $this->dbxClient->createFolder($path);
264 
265         return $result;
266     }
267 
268     /**
269     * getDelta
270     *
271     * Liefert alle Änderungen des virtuellen Datenspeichers seit dem übergebenen Cursor
272     * Wird null als Cusor übergeben werden alle Änderungen seit bestehen des virtuellen
Datenspeichers zurückgeleifert.
273     *
274     * @param string     $delta value        Cursor ab dem das Delta berechnet werden
soll
275     *
276     * @return Array     Alle Änderungen des Datenspeichers ab dem übergebenen Cursor
277     *                   Struktur:
278     *                   $ret["cursor"]= Name von $file
279     *                   $ret["entries"][] = Eindeutiger Bezeichnung des virtuellen
Datenspeichers
280     *                   $ret["entries"][]["path"] = Pfad zur/zum betroffenen
Datei/Ordner
281     *                   $ret["entries"][]["action"] = Durchgeführte Aktion
[delete|newdir|upload]
282     */
283     function getDelta($delta_value){
284         $result = $this->dbxClient->getDelta($delta_value,null);
285 
286         //einheitliches Return Array aufbauen
287         $ret["cursor"] = $result["cursor"];
288         foreach ($result["entries"] as $key => $value) {
289             if(strtolower(substr($value[0],0,strlen($this-
>ext_root)+1))==strtolower($this->ext_root."/")){
290                 $ret["entries"][$key]["path"] = substr($value[0],strlen($this-
>ext_root)+1);
291                 if(sizeof($value[1])==0)
292                     $ret["entries"][$key]["action"]="delete";
293                 elseif($value[1]["is_dir"]==1)
294                     $ret["entries"][$key]["action"] = "newdir";
295                 else
296                     $ret["entries"][$key]["action"] = "upload";
297             }
298         }
299         //$ret["debug"] = $result;
300         return $ret;
301     }
302 
303 }
304 
305 ?>
```

D.11. Klasse Datastorage_Googledrive: classes/Datastorage_Googledrive.php

```
001 <?php
002
003 require_once "classes/Datastorage.php";
004 set_include_path("api/google-api-php-client/src" . PATH_SEPARATOR . get_include_path());
005 require_once 'Google/Client.php';
006 require_once 'Google/Service/Drive.php';
007 require_once 'Google/Http/MediaFileUpload.php';
008 require_once 'Google/Http/Request.php';
009
010 /**
011  * Datastorage Googledrive
012  *
013  * Klasse für die Instanzierung von Google Drive Speichern als virtuelle Datenspeicher
des Cloud Storage Pools (CSP)
014  *
015  * @author      Peter Völkl <peter@vape.net>
016  * @package     CSP
017  * @version     2014-04-26
018  */
019 class Datastorage_Googledrive extends Datastorage{
020     private $gdrClient;
021     private $gdrClient_client;
022
023     /**
024     * __construct
025     *
026     * Initialisierung der allgemenen Klassenvariablen
027     *
028     * @param string     $id         Eindeutige ID des virtuellen Datenspeichers
029     * @param string     $name       Eindeutiger Bezeichnung des virtuellen
Datenspeichers
030     * @param string     $ext_root   Root Ordner der Instanz innerhalb des externen
Datenspeichers
031     * @param string     $token      Authorisierungs Token zur Verbindung mit dem
externen Datnespeicher
032     * @param string     $crypto key Passphrase zur Verschlüsselung der Daten innderhalb
des externen Datenspeichers
033     * @param Array      $appinfo    Dropbox Appinfo Array("key","secret")
034     */
035     function __construct($id,$name,$ext_root,$token,$crypto_key,$appinfo){
036         parent::__construct($id,$name,$ext_root,$token,$crypto_key);
037         $this->type = "GOODR";
038
039         //Google Drive Zugriff initialisieren
040         $client = new Google_Client();
041         $client->setClientId($appinfo["client id"]);
042         $client->setClientSecret($appinfo["client secret"]);
043         $client->addScope("https://www.googleapis.com/auth/drive");
044         $client->setAccessToken($token);
045
046         $this->gdrClient client = $client;
047         $this->gdrClient = new Google_Service_Drive($client);
048     }
049
050     /**
051     * getMetadataByPath
052     *
053     * Liefert die Metadaten einer Google Drive Datei anhand ihres Dateipfades zurück
054     *
055     * @param string     $path   Dateipfad
056     *
057     * @return Google DriveFile  Metadaten der Datei
058     *                     https://developers.google.com/drive/v2/reference/files#resource
059     */
060     private function getMetadataByPath($path){
061         if(substr($path,0,1)=="/")
062             $path = substr($path,1);
063         $path_explode = explode("/", $path);
064         $last id["id"] = "root";
065
066         //Suche der FileID in Google Drive Struktur
```

```
067         foreach ($path_explode as $key => $value) {
068             //Prüfen ob vorletztes Element des Pfades erreicht ist
069             $parameters = array();
070             $parameters["q"] = "'".$last_id["id"]."' in parents and title =
'".$value."'";
071 
072             try{
073                 $files = $this->gdrClient->files->listFiles($parameters);
074             } catch(Exception $e) {
075                 $last_id=null;
076                 break;
077             }
078 
079             if(get class($files)=="Google_Service_Drive_FileList" &&
count($files["modelData"]["items"]) > 0){
080                 $last id = $files["modelData"]["items"]["0"];
081             } else {
082                 $last_id=null;
083                 break;
084             }
085         }
086 
087         return $last_id;
088     }
089 
090     /**
091     * getPathById
092     *
093     * Liefert den Dateipfad einer Google Drive Datei zurück
094     *
095     * @param string     $id     FileID der Google Drive Datei
096     *
097     * @return string    Dateipfad
098     */
099     function getPathById($id){
100         $last_id = $id;
101         $path = "";
102 
103         //Suche der FileID in Google Drive Struktur
104         while($last_id) {
105             //Prüfen ob vorletztes Element des Pfades erreicht ist
106             $parameters = array();
107             $parameters["q"] = "id = '".$last_id."'";
108 
109             try{
110                 $file = $this->gdrClient->files->get($last id);
111             } catch(Exception $e) {
112                 $last_id=null;
113                 break;
114             }
115 
116             if(count($file["modelData"]) > 0
117                 && count($file["modelData"]["parents"]) > 0){
118                 $last_id = $file["modelData"]["parents"]["0"]["id"];
119                 if(strlen($path)>0)
120                     $path = "/".$file["title"].$path;
121                 else
122                     $path = "/".$file["title"];
123             } else {
124                 $last_id=null;
125                 break;
126             }
127         }
128 
129         return $path;
130     }
131 
132     /**
133     * getFilelist
134     *
135     * Abfrage aller Dateien und ihrer Eigenschaften eines Verzeichnisses innerhalb des
virtuellen Datenspeichers
136     *
137     * @param string     $directory      Verzeichnispfad
```

```
138     *
139     * @return Array     Daten des Ordners oder der Datei $directory inklusive aller
enthaltenen Dateien und Ordner
140     *                   Struktur:
141     *                   $ret["name"]= Name von $directory
142     *                   $ret["path"] = Pfad innerhalb des virtuellen Datenspeichers zu
$directory
143     *                   $ret["datastorage_name"] = Eindeutiger Bezeichnung des
virtuellen Datenspeichers
144     *                   $ret["creation date"] = Erstellungsdatum von $directory;
145     *                   $ret["last modified"] = Datum der letzten Änderung von
$directory;
146     *                   $ret["is dir"] = [1...wenn es sich um ein Verzeichnis handelt]
147     *                   $ret["mime_type"] = MIME Typ von $directory
148     *                   $ret["bytes"] = Größe von $directory in Byte
149     *                   $ret["contents"][] = Dateien und Ordner innrehalb von $directory
150     *                   $ret["contents"][]["name"]= Name des Elements
151     *                   $ret["contents"][]["path"] = Pfad innerhalb des virtuellen
Datenspeichers zum Element
152     *                   $ret["contents"][]["datastorage_name"] = Eindeutiger Bezeichnung
des virtuellen Datenspeichers
153     *                   $ret["contents"][]["creation date"] = Erstellungsdatum des
Elements;
154     *                   $ret["contents"][]["last modified"] = Datum der letzten Änderung
des Elements;
155     *                   $ret["contents"][]["is_dir"] = [1...wenn es sich um ein
Verzeichnis handelt]
156     *                   $ret["contents"][]["mime type"] = MIME Typ des Elements
157     *                   $ret["contents"][]["bytes"] = Größe des Elements in Byte
158     */
159     function getFilelist($directory){
160         $ret = null;
161
162         if(strlen($directory)>0)
163             $path=$this->ext root."/".$directory;
164         else
165             $path=$this->ext_root;
166
167         $folderMetadata = $this->getMetadataByPath($path);
168
169         if($folderMetadata){
170             $ret = array();
171             $ret["contents"] = Array();
172             $ret["name"]= basename($directory);
173             $ret["path"] = substr($directory, 0, strlen($directory)-
strlen(basename($directory)));
174             if(substr($ret["path"],-1)=="/")
175                 $ret["path"] = substr($ret["path"],0,-1);
176             $ret["datastorage_name"]=$this->name;
177             $ret["creation date"] = strtotime($folderMetadata["createdDate"]);
178             $ret["last modified"] = strtotime($folderMetadata["modifiedDate"]);
179             $ret["is_dir"] = ($folderMetadata["mimeType"]=="application/vnd.google-
apps.folder"?"1":"0");
180
181             //Ordnerinhalt abrufen
182             $parameters = array();
183             $parameters['q'] = "'".$folderMetadata["id"]."' in parents";
184             try{
185                 $files = $this->gdrClient->files->listFiles($parameters);
186             } catch(Exception $e) {
187                 break;
188             }
189
190             if(isset($files) && count($files["modelData"]["items"])>0){
191                 foreach ($files["modelData"]["items"] as $key => $value) {
192                     if(!isset($value["explicitlyTrashed"]) ||
$value["explicitlyTrashed"]!=1){
193                         $ret["contents"][$key]["name"] = $value["title"];
194                         $ret["contents"][$key]["path"] = $directory;
195                         $ret["contents"][$key]["datastorage_name"]=$this->name;
196                         $ret["contents"][$key]["is_dir"] =
($value["mimeType"]=="application/vnd.google-apps.folder"?"1":"0");
197                         $ret["contents"][$key]["creation date"] =
strtotime($value["createdDate"]);
```

```
198                         $ret["contents"][$key]["last modified"] =
strtotime($value["modifiedDate"]);
199                         if(isset($value["mimeType"]))
200                             $ret["contents"][$key]["mime_type"] = $value["mimeType"];
201                         if(isset($value["fileSize"]))
202                             $ret["contents"][$key]["bytes"] = $value["fileSize"];
203                     }
204                 }
205             }
206         }
207
208         return $ret;
209     }
210
211     /**
212     * getFile
213     *
214     * Liefert eine Datei und ihre Eigenschaften zurück
215     *
216     * @param string     $file       Pfad innerhalb des virtuellen Datenspeichers zur
Datei
217     *
218     * @return Array     Eigenschaften der Datei und Pfad zu temporärer Datei für ihren
Abruf
219     *                   Struktur:
220     *                   $ret["name"]= Name von $file
221     *                   $ret["datastorage name"] = Eindeutiger Bezeichnung des
virtuellen Datenspeichers
222     *                   $ret["creation date"] = Erstellungsdatum von $file;
223     *                   $ret["last_modified"] = Datum der letzten Änderung von $file;
224     *                   $ret["is_dir"] = [1...wenn es sich um ein Verzeichnis handelt]
225     *                   $ret["mime_type"] = MIME Typ von $directory
226     *                   $ret["bytes"] = Größe von $directory in Byte
227     *                   $ret["tmpfile"] = Pfad zur angelegten temporärern Datei für
weitere Verarbeitung
228     */
229     function getFile($file){
230
231         $local_temp = "tmp/";
232
233
234
235         if($file=="")
236             $path=$this->ext_root;
237         elseif(substr($file,0,1)=="/")
238             $path=$this->ext_root.$file;
239         else
240             $path=$this->ext_root."/".$file;
241
242         $filename=basename($path);
243
244         $fileMetadata = $this->getMetadataByPath($path);
245         $fileList=null;
246         if(isset($fileMetadata) && $fileMetadata != null
247             && $fileMetadata["mimeType"]!="application/vnd.google-apps.folder"){
248             //Ist Datei
249
250             $fp = fopen($local_temp.$filename, "w+b");
251
252             //Entschlüsselung
253             if($this->crypto_key != null){
254                 $this->csp_decryptStream($fp,$this->crypto_key);
255             }
256
257             try{
258                 $file = $this->gdrClient->files->get($fileMetadata["id"]);
259                 $downloadUrl = $file->getDownloadUrl();
260                 if ($downloadUrl) {
261                   $request = new Google_Http_Request($downloadUrl, 'GET', null, null);
262                   $httpRequest = $this->gdrClient_client->getAuth()-
>authenticatedRequest($request);
263
264                   if ($httpRequest->getResponseHttpCode() == 200) {
265                     fwrite($fp,$httpRequest->getResponseBody());
```

```
266                     }
267                 }
268                 $ret["name"]=$filename;
269                 $ret["datastorage_name"]=$this->name;
270                 $ret["tmpfile"]=$local_temp.$filename;
271                 $ret["mime type"]=$fileMetadata["mimeType"];
272                 $ret["bytes"]=$fileMetadata["fileSize"];
273                 $ret["creation_date"] = strtotime($fileMetadata["createdDate"]);
274                 $ret["last_modified"] = strtotime($fileMetadata["modifiedDate"]);
275                 $ret["is dir"] = 0;
276             }catch(Exception $e){
277                 //Fehler beim Download
278                 $ret=null;
279             }
280 
281             fclose($fp);
282 
283         } elseif(($fileList = $this->getFilelist($file)) && $fileList["is_dir"]==1){
284             //Ist Verzeichnis
285             $ret["name"]=$filename;
286             $ret["datastorage_name"]=$this->ext_root;
287             $ret["mime_type"]=null;
288             $ret["bytes"]=0;
289             $ret["creation date"] = strtotime($fileMetadata["createdDate"]);
290             $ret["last modified"] = strtotime($fileMetadata["modifiedDate"]);
291             if(isset($fileList["contents"]))
292                 $ret["file list"] = $fileList["contents"];
293             $ret["is dir"] = 1;
294         } else {
295             $ret = null;
296         }
297 
298         return $ret;
299     }
300 
301     /**
302     * uploadFile
303     *
304     * Ladet eine Datei in den virtuellen Datenspeicher hoch
305     *
306     * @param string     $tmpfile        Pfad zur temporären Datei die hochgeladen werden
soll
307     * @param string     $newfile        Pfad und Name der neuen Datei innerhalb des
virtuellen Datenspeichers
308     */
309     function uploadFile($tmpfile,$newfile){
310 
311         if($newfile=="")
312             return false;
313         elseif(substr($newfile,0,1)=="/")
314             $path=$this->ext root.$newfile;
315         else
316             $path=$this->ext_root."/".$newfile;
317 
318         //Wenn Datei existiert wird ihre id zwischengespeichert
319         //und nach dem neuen Upload gelöscht
320         $oldFileMetadata = $this->getMetadataByPath($path);
321         $folderMetadata = $this->getMetadataByPath(dirname($path));
322 
323         if($folderMetadata){
324             $parentId = $folderMetadata["id"];
325 
326             $file = new Google_Service_Drive_DriveFile();
327             $file->setTitle(basename($path));
328             $parent = new Google_Service_Drive_ParentReference();
329             $parent->setId($parentId);
330             $file->setParents(array($parent));
331 
332             $fp = fopen($tmpfile, "rb");
333 
334             //Verschlüsselung
335             if($this->crypto_key != null){
336                 $this->csp encryptStream($fp,$this->crypto key);
337             }
```

```
338
339             $data = stream_get_contents($fp);
340
341             $result = $this->gdrClient->files->insert($file,array(
342               'data' => $data,
343               'uploadType' => 'media'
344             ));
345
346             fclose($fp);
347
348             //Wenn eine alte version existiert, wird sie gelöscht
349             if($oldFileMetadata){
350                 try{
351                     $this->gdrClient->files->delete($oldFileMetadata["id"]);
352                 } catch(Exception $e){
353                     //Fehler beim löschen
354                     //beide Dateien bleiben bestehen
355                 }
356             }
357
358             return $result;
359         }
360         return null;
361     }
362
363
364     /**
365     * uploadFileStream
366     *
367     * Ladet eine Datei in den virtuellen Datenspeicher hoch
368     *
369     * @param string     $fp              Filestream der hochzuladenden Datei
370     * @param string     $newfile         Pfad und Name der neuen Datei innerhalb des
virtuellen Datenspeichers
371     */
372     function uploadFileStream($fp,$newfile){
373         if($newfile=="")
374             return false;
375         elseif(substr($newfile,0,1)=="/")
376             $path=$this->ext root.$newfile;
377         else
378             $path=$this->ext_root."/".$newfile;
379
380         //Wenn Datei existiert wird ihre id zwischengespeichert
381         //und nach dem neuen Upload gelöscht
382         $oldFileMetadata = $this->getMetadataByPath($path);
383
384         $folderMetadata = $this->getMetadataByPath(dirname($path));
385
386         if($folderMetadata){
387             $parentId = $folderMetadata["id"];
388
389             $file = new Google_Service_Drive_DriveFile();
390             $file->setTitle(basename($path));
391             $parent = new Google_Service_Drive_ParentReference();
392             $parent->setId($parentId);
393             $file->setParents(array($parent));
394
395             //Verschlüsselung
396             if($this->crypto_key != null){
397                 $this->csp encryptStream($fp,$this->crypto key);
398             }
399
400             $data = stream_get_contents($fp);
401
402             $result = $this->gdrClient->files->insert($file,array(
403               'data' => $data,
404               'uploadType' => 'media'
405             ));
406
407             fclose($fp);
408
409             //Wenn eine alte version existiert, wird sie gelöscht
410             if($oldFileMetadata){
```

```
411                 try{
412                     $this->gdrClient->files->delete($oldFileMetadata["id"]);
413                 } catch(Exception $e){
414                     //Fehler beim Löschen
415                     //beide Dateien bleiben bestehen
416                 }
417             }
418 
419             return $result;
420         }
421         return null;
422     }
423 
424     /**
425     * deleteFile
426     *
427     * Löscht eine Datei des virtuellen Datenspeichers
428     *
429     * @param string     $file       Pfad innerhalb des virtuellen Datenspeichers zur
Datei
430     */
431     function deleteFile($file){
432         if(substr($file,0,1)=="/")
433             $file = substr($file,1);
434 
435         $path=$this->ext_root."/".$file;
436 
437         $fileMetadata = $this->getMetadataByPath($path);
438 
439         try{
440             $result = $this->gdrClient->files->delete($fileMetadata["id"]);
441         }catch(Exception $e){
442             //Fehler beim Löschen
443             return null;
444         }
445 
446         return $result;
447     }
448 
449     /**
450     * copyFile
451     *
452     * Kopiert eine Datei innerhalb des Datenspeichers
453     *
454     * @param String     $source file    Pfad der Quelldatei innerhalb des virtuellen
Datenspeichers
455     * @param String     $dest file  Pfad der Quelldatei innerhalb des virtuellen
Datenspeichers
456     */
457     function copyFile($source_file, $dest_file){
458         if(substr($source_file,0,1)=="/")
459             $source file = substr($source_file,1);
460         if(substr($dest_file,0,1)=="/")
461             $dest_file = substr($dest_file,1);
462 
463         $source file path=$this->ext root."/".$source file;
464         $dest file path=$this->ext root."/".$dest file;
465 
466         $fileMetadata_source = $this->getMetadataByPath($source_file_path);
467         $fileMetadata_destdir = $this->getMetadataByPath(dirname($dest_file_path));
468         if($fileMetadata_destdir && $fileMetadata_source){
469             $parentId = $fileMetadata destdir["id"];
470             $originFileId = $fileMetadata source["id"];
471 
472             $copiedFile = new Google_Service_Drive_DriveFile();
473             $copiedFile->setTitle(basename($dest file path));
474             $parent = new Google_Service_Drive_ParentReference();
475             $parent->setId($parentId);
476             $copiedFile->setParents(array($parent));
477             try {
478               return $this->gdrClient->files->copy($originFileId, $copiedFile);
479             } catch (Exception $e) {
480               return null;
481             }
```

```
482         }
483
484         return null;
485     }
486
487     /**
488     * createFolder
489     *
490     * Erstellt einen neuen Ordner innerhalb des virtuellen Datenspeichers
491     *
492     * @param string     $folder     Pfad des neuen Ordners inkusive Ordnername innerhalb
des virtuellen Datenspeichers
493     */
494     function createFolder($folder){
495         if($folder=="")
496             return false;
497         elseif(substr($folder,0,1)=="/")
498             $path=$this->ext root.$folder;
499         else
500             $path=$this->ext_root."/".$folder;
501
502         $newFolderMetadata = $this->getMetadataByPath($path);
503         $folderMetadata = $this->getMetadataByPath(dirname($path));
504
505         //Ordner nur Anlegen wenn übergordneter Ordner exisiert und er selbst noch nicht
exisiert
506         if($folderMetadata && !$newFolderMetadata){
507             $parentId = $folderMetadata["id"];
508
509             $file = new Google_Service_Drive_DriveFile();
510             $file->setTitle(basename($path));
511             $parent = new Google_Service_Drive_ParentReference();
512             $parent->setId($parentId);
513             $file->setParents(array($parent));
514             $file->setMimetype("application/vnd.google-apps.folder");
515
516             return $this->gdrClient->files->insert($file);
517
518         }
519         return null;
520     }
521
522     /**
523     * getDelta
524     *
525     * Liefert alle Änderungen des virtuellen Datenspeichers seit dem übergebenen Cursor
526     * Wird null als Cusor übergeben werden alle Änderungen seit bestehen des virtuellen
Datenspeichers zurückgeleifert.
527     *
528     * @param string     $delta value        Cursor ab dem das Delta berechnet werden
soll
529     *
530     * @return Array     Alle Änderungen des Datenspeichers ab dem übergebenen Cursor
531     *                   Struktur:
532     *                   $ret["cursor"]= Name von $file
533     *                   $ret["entries"][] = Eindeutiger Bezeichnung des virtuellen
Datenspeichers
534     *                   $ret["entries"][]["path"] = Pfad zur/zum betroffenen
Datei/Ordner
535     *                   $ret["entries"][]["action"] = Durchgeführte Aktion
[delete|newdir|upload]
536     */
537     function getDelta($delta_value){
538
539         $pageToken = null;
540         if($delta_value)
541             $startChangeId = $delta value;
542         else
543             $startChangeId = null;
544         $result = Array();
545
546         do {
547           try {
548             $parameters = array();
```

```
549             if ($startChangeId) {
550               $parameters['startChangeId'] = $startChangeId;
551             }
552             if ($pageToken) {
553               $parameters['pageToken'] = $pageToken;
554             }
555             $parameters["includeSubscribed"] = false;
556             $changes = $this->gdrClient->changes->listChanges($parameters);
557             $result = array_merge($result, $changes->getItems());
558             $pageToken = $changes->getNextPageToken();
559           } catch (Exception $e) {
560             print "An error occurred: " . $e->getMessage();
561             $pageToken = NULL;
562           }
563         } while ($pageToken);
564 
565         $ret["cursor"] = $startChangeId;
566         $ret["entries"] = Array();
567 
568         //Array mit Updates aufbauen
569         foreach ($result as $key => $value) {
570 
571             //Erstes Elemant überspringen, da der Cursor auf einen bereits geprüften
Eintrag zeigt
572             if($key==0)
573                 continue;
574 
575             $path =  $this->getPathById($value["fileId"]);
576             $path_explode = explode("/", $path);
577 
578             $ret["cursor"] = $value["id"];
579             if(count($path_explode)>0
580                 && substr($path,0,strlen($this->ext root)+1)==($this->ext root."/")
581                 && $path != $this->ext root){
582                 $new_entry = Array();
583                 $new_entry["path"] = substr($path,strlen($this->ext_root)+1);
584 
585                 if( (sizeof($value["modelData"])==0 && $value["deleted"]==1)
586                     || ($value["modelData"]["file"]["labels"]["trashed"]==1) ){
587                     $new entry["action"]="delete";
588                 }elseif(isset($value["modelData"]["file"])){
589 
if($value["modelData"]["file"]["mimeType"]=="application/vnd.google-apps.folder")
590                             $new_entry["action"] = "newdir";
591                         else
592                             $new entry["action"] = "upload";
593                 }
594                 if(isset($new_entry["action"]))
595                     $ret["entries"][] = $new_entry;
596             }
597 
598         }
599         return $ret;
600     }
601 
602 }
603 
604 ?>
```

D.12. Klasse CspWebdav: classes/CspWebdav.php

```
001 <?php
002
003     require_once "config.inc.php";
004     require_once "functions.inc.php";
005     require_once "classes/Datastorage.php";
006     require_once "classes/Datastorage_Dropbox.php";
007     require_once "classes/Datastorage_Googledrive.php";
008     require_once "api/HTTP_WebDAV_Server/Server.php";
009
010     /**
011      * CspWebdav
012      *
013      * WebDAV Schnittstelle für den Zugriff auf die virtuellen Datenspeicher des Cloud
Storage Pools (CSP)
014      * Ableitung der abstrakten Klasse HTTP_WebDAV_Server und basierend auf dem
Codebeispiel HTTP_WebDAV_Server_Filesystem von Hartmut Holzgraefe
015      *
016      * @author      Peter Völkl <peter@vape.net>
017      * @package     CSP
018      * @version     2014-04-26
019      */
020     class CspWebdav extends HTTP_WebDAV_Server{
021
022         private $config_appinfos;
023         private $tmp_files = Array();
024
025         /**
026         * setConfigAppinfos
027         *
028         * Stellt das Array mit den Appinfos der externen Datenspeicher zur Verfügung
029         *
030         * @param Array  $config_appinfos        Appinfo Array
031         */
032         function setConfigAppinfos($config_appinfos){
033             $this->config appinfos = $config appinfos;
034         }
035
036
037         /**
038         * ServeRequest
039         *
040         * Verarbeitet die WebDAV Anfrage
041         *
042         */
043         function ServeRequest()
044         {
045             //Startet den WebDAV Server und verarbeitet die Anfrage
046             parent::ServeRequest();
047
048             //Temp Dateien löschen (die bei GET entstanden sind)
049             foreach ($this->tmp_files as $key => $value) {
050                 unlink($value);
051             }
052         }
053
054         /**
055         * check auth
056         *
057         * Führt die Prüfung der Authentizität durch.
058         * Für den CSP nicht implementiert, da über Basic-Auth mittels .htaccess
gearbeitet wird
059         *
060         * @param String     $type       Authentifikationtyp
061         * @param String     $user       Username
062         * @param String     $pass       Passwort
063         */
064         function check_auth($type, $user, $pass)
065         {
066             return true;
067         }
068
```

```
069         /**
070         * PROPFIND
071         *
072         * Implementierung der Methode PROPFIND des WebDAV Servers
073         *
074         * @param Array      &$options
075         * @param Array      &$files
076         */
077         function PROPFIND(&$options, &$files)
078         {
079             $options["path"] = urldecode($options["path"]);
080 
081             //Prüfen ob Anfrage in einem Datastorage oder im Root ausgeführt wurde
082             if($options["path"]=="" || $options["path"]=="/"){
083                 //Angefragter Pfad bezieht sich auf Root-Verzeichnis
084                 //Liste der Datastorages zurückgeben
085                 $res = mysql_query("select id,name from csp_datastorages
086                     where primary storage id is null
087                     order by name
088                     ");
089                 $files["files"] = array();
090 
091                 //Rootverzeichnis
092                 $info = array();
093                 $info["path"]  = "/";
094                 $info["props"] = array();
095                 $info["props"][] = $this->mkprop("displayname",      "");
096                 $info["props"][] = $this->mkprop("creationdate",     time());
097                 $info["props"][] = $this->mkprop("getlastmodified", time());
098                 $info["props"][] = $this->mkprop("resourcetype", "collection");
099                 $info["props"][] = $this->mkprop("getcontenttype", "httpd/unix-
directory");
100                 //Microsoft-spezifisch:
101                 $info["props"][] = $this->mkprop("lastaccessed",     time());
102                 $info["props"][] = $this->mkprop("ishidden",         false);
103 
104                 $files["files"][] = $info;
105 
106                 while($arr=mysql_fetch_array($res)){
107 
108                     $info = array();
109                     $info["path"]  = "/".$arr["name"];
110                     $info["props"] = array();
111                     $info["props"][] = $this->mkprop("displayname",      $arr["name"]);
112                     $info["props"][] = $this->mkprop("creationdate",     time());
113                     $info["props"][] = $this->mkprop("getlastmodified", time());
114                     $info["props"][] = $this->mkprop("resourcetype", "collection");
115                     $info["props"][] = $this->mkprop("getcontenttype", "httpd/unix-
directory");
116                     //Microsoft-spezifisch:
117                     $info["props"][] = $this->mkprop("lastaccessed",     time());
118                     $info["props"][] = $this->mkprop("ishidden",         false);
119 
120                     $files["files"][] = $info;
121                 }
122                 return true;
123 
124             }else{
125                 //Angefragter Pfad befindet sich in einem Datastorage
126 
127                 $path_explode = explode("/",$options["path"]);
128                 $ds path = substr($options["path"],strlen($path explode[1])+2);
129                 if(substr($ds_path,-1)=="/")
130                     $ds_path=substr($ds_path,0,-1);
131 
132                 //Parameter des Datastorages laden
133                 $ds = getDatastorageByName($path explode[1],$this->config appinfos);
134                 if($ds){
135                     $files["files"] = array();
136                     $filedata = $ds->getFilelist($ds_path);
137 
138                     if(!isset($filedata["name"]))
139                         return false;
140 
```

```
                    //Die Information über das Verzeichnis selbst hinzufügen
                    $files["files"][] = $this->fileinfo($filedata);

                    //Die Information der Dateien im Verzeichnis hinzufügen
                    if(isset($filedata["contents"])){
                        foreach ($filedata["contents"] as $key => $value) {
                            $files["files"][] = $this->fileinfo($value);
                        }
                    }
                    return true;
                } else {
                    return false;
                }
            }
            return false;

        }

        /**
        * fileinfo
        *
        * Wandelt ein Array mit Dateieigenschaften eines Datastorages in ein Array für
den WebDAV Server um.
        *
        * @param Array      Array mit den Dateieigenschaften eines Datastorages
        *
        * @param Array      Array mit den Dateieigenschaften für den WebDAV Server
        */
        private function fileinfo($file) {
            $info = array();
            if(isset($file["datastorage_name"]) && $file["datastorage_name"]!="")
                $info["path"]  =
"/".$file["datastorage name"]."/".($file["path"]!=""?$file["path"]."/":"").$file["name"];
            else
                $info["path"]  = "/".$file["name"];
            $info["props"] = array();

            $info["props"][] = $this->mkprop("displayname",      $file["name"]);
            $info["props"][] = $this->mkprop("creationdate",     $file["creation date"]);
            $info["props"][] = $this->mkprop("getlastmodified", $file["last modified"]);
            if($file["is_dir"]==1){
                $info["props"][] = $this->mkprop("resourcetype", "collection");
                $info["props"][] = $this->mkprop("getcontenttype", "httpd/unix-
directory");
            }else{
                $info["props"][] = $this->mkprop("resourcetype",     "");
                $info["props"][] = $this->mkprop("getcontenttype",
isset($file["mime_type"])?$file["mime_type"]:"");
                $info["props"][] = $this-
>mkprop("getcontentlength",isset($file["bytes"])?$file["bytes"]:"");
            }
            //Microsoft-spezifisch:
            $info["props"][] = $this->mkprop("lastaccessed",     $file["last_modified"]);
            $info["props"][] = $this->mkprop("ishidden",         false);

            //$this->optfileinfo($info["props"]);
            return $info;
        }

        /**
        * GET
        *
        * Implementierung der Methode GET des WebDAV Servers
        *
        * @param Array      &$options
        */
        function GET(&$options)
        {
            $options["path"] = urldecode($options["path"]);

            //Parameter des Datastorages laden
            $path_explode = explode("/",$options["path"]);
            $ds path = substr($options["path"],strlen($path explode[1])+2);
            if(substr($ds_path,-1)=="/")
```

```
210                 $ds path=substr($ds path,0,-1);
211
212             //Parameter des Datastorages laden
213             $ds = getDatastorageByName($path_explode[1],$this->config_appinfos);
214             if($ds){
215                 $filedata = $ds->getFile($ds path);
216                 if($filedata == null){
217
218                 } elseif($filedata["is_dir"]){
219                     return $this->GetDir($filedata, $options);
220                 }else{
221                     $options['stream'] = fopen($filedata["tmpfile"], "r");
222                     $this->tmp files[] = $filedata["tmpfile"];
223                     return true;
224                 }
225             } else {
226                 return false;
227             }
228             return false;
229         }
230
231         /**
232         * GetDir
233         *
234         * Gibt eine HTML formatierte Liste des Inhalts eines Verzeichnisses aus
235         *
236         * @param Array      &$filedata  Ordnerdaten und Dateiliste (Datastorage-
>getFilelist())
237         * @param Array      &$options   Serveroptionen
238         */
239         function GetDir($filedata, &$options)
240         {
241             $format = "%15s  %-19s  %-s\n";
242
243             echo "<html><head><title>Index of
".htmlspecialchars($options['path'])."</title></head>\n";
244
245             echo "<h1>Index of ".htmlspecialchars($options['path'])."</h1>\n";
246
247             echo "<pre>";
248             printf($format, "Size", "Last modified", "Filename");
249             echo "<hr>";
250             $path explode = explode("/",$options['path']);
251             $path = $path_explode[count($path_explode)-1];
252             foreach ($filedata["file_list"] as $key => $value) {
253                 $fullpath = $path."/".$value["name"];
254                 $name     = htmlspecialchars($value["name"]);
255                 printf($format,
256                        number_format(filesize($fullpath)),
257                        strftime("%Y-%m-%d %H:%M:%S", filemtime($fullpath)),
258                        '<a href="' . $fullpath . '">' . $name . '</a>');
259             }
260             echo "</pre>\n";
261
262             echo "</html>\n";
263
264             exit;
265         }
266
267         /**
268         * PUT
269         *
270         * Implementierung der Methode PUT des WebDAV Servers
271         *
272         * @param Array      &$options
273         * @param Array      &$files
274         */
275         function PUT(&$options)
276         {
277             $options["path"] = urldecode($options["path"]);
278
279             //Parameter des Datastorages laden
280             $path_explode = explode("/",$options["path"]);
281             $ds_path = substr($options["path"],strlen($path_explode[1])+2);
```

```
282             if(substr($ds_path,-1)=="/")
283                 $ds_path=substr($ds_path,0,-1);
284 
285             //Parameter des Datastorages laden
286             $ds = getDatastorageByName($path_explode[1],$this->config_appinfos);
287             if($ds){
288                     //Dateiupload durchführen
289                     $result = $ds->uploadFileStream($options["stream"],$ds_path);
290 
291                     return "";
292             } else {
293                 return "403 Forbidden";
294             }
295             return "403 Forbidden";
296         }
297 
298         /**
299         * MKCOL
300         *
301         * Implementierung der Methode MKCOL des WebDAV Servers
302         *
303         * @param Array      &$options
304         * @param Array      &$files
305         */
306         function MKCOL($options)
307         {
308             $options["path"] = urldecode($options["path"]);
309 
310             //Parameter des Datastorages laden
311             $path_explode = explode("/",$options["path"]);
312             $ds path = substr($options["path"],strlen($path_explode[1])+2);
313             if(substr($ds_path,-1)=="/")
314                 $ds path=substr($ds path,0,-1);
315 
316             //Parameter des Datastorages laden
317             $ds = getDatastorageByName($path_explode[1],$this->config_appinfos);
318             if($ds){
319                 //Ordner anlegen
320                 $result = $ds->createFolder($ds path);
321 
322                 return ("201 Created");
323             } else {
324                 return "403 Forbidden";
325             }
326             return "403 Forbidden";
327         }
328 
329         /**
330         * DELETE
331         *
332         * Implementierung der Methode DELETE des WebDAV Servers
333         *
334         * @param Array      &$options
335         * @param Array      &$files
336         */
337         function DELETE($options)
338         {
339             $options["path"] = urldecode($options["path"]);
340 
341             //Parameter des Datastorages laden
342             $path_explode = explode("/",$options["path"]);
343             $ds path = substr($options["path"],strlen($path explode[1])+2);
344             if(substr($ds_path,-1)=="/")
345                 $ds_path=substr($ds_path,0,-1);
346 
347             //Parameter des Datastorages laden
348             $ds = getDatastorageByName($path explode[1],$this->config appinfos);
349             if($ds){
350                 //Datei/Ordner löschen
351                 $result = $ds->deleteFile($ds_path);
352 
353                 return "204 No Content";
354             } else {
355                 return "404 Not found";
```

```
356             }
357             return "404 Not found";
358         }
359 
360         /**
361         * MOVE
362         *
363         * Implementierung der Methode MOVE des WebDAV Servers
364         *
365         * @param Array      &$options
366         * @param Array      &$files
367         */
368         function MOVE($options)
369         {
370             return $this->COPY($options, true);
371         }
372 
373         /**
374         * COPY
375         *
376         * Implementierung der Methode COPY des WebDAV Servers
377         *
378         * @param Array      &$options
379         * @param Array      &$files
380         */
381         function COPY($options, $del=false)
382         {
383 
384             if (!empty($this->_SERVER["CONTENT_LENGTH"])) { // no body parsing yet
385                 return "415 Unsupported media type";
386             }
387             // no copying to different WebDAV Servers yet
388             if (isset($options["dest_url"])) {
389                 return "502 bad gateway";
390             }
391 
392 
393             $source = urldecode($options["path"]);
394             $dest = urldecode($options["dest"]);
395 
396 
397             //Parameter des Datastorages laden
398             $source explode = explode("/",$source);
399             $dest_explode = explode("/",$dest);
400 
401             //Prüfen ob das Kopieren innerhalb eines Datastorages geschieht
402             if($source_explode[1] == $dest_explode[1]){
403                 //Datastoreage von Quelle und Ziel sind ident
404 
405                 $ds source path = substr($source,strlen($source explode[1])+2);
406                 if(substr($ds_source_path,-1)=="/")
407                     $ds_source_path=substr($ds_source_path,0,-1);
408 
409                 $ds_dest_path = substr($dest,strlen($dest_explode[1])+2);
410                 if(substr($ds_dest_path,-1)=="/")
411                     $ds dest path=substr($ds dest path,0,-1);
412 
413                 //Parameter des Datastorages laden
414                 $ds = getDatastorageByName($source_explode[1],$this->config_appinfos);
415                 if($ds){
416                     //Datei/Ordner kopieren
417                     $ds->copyFile($ds source path,$ds dest path);
418 
419                     if($del)
420                         $ds->deleteFile($ds_source_path);
421 
422                     return "201 Created";
423                 } else {
424                     return "404 Not found";
425                 }
426                 return "404 Not found";
427 
428             }else{
429                 //Datastoreage von Quelle und Ziel sind unterschiedlich
```

```
430
431                 //Nicht zulässig
432                 return "204 No Content";
433             }
434
435             return "404 Not found";
436         }
437
438         /**
439         * PROPPATCH
440         *
441         * Implementierung der Methode PROPPATCH des WebDAV Servers
442         * Eine Änderungen von Dateieigenschaften ist derzeit nicht vorgesehen
443         *
444         * @param Array      &$options
445         * @param Array      &$files
446         */
447         function PROPPATCH(&$options)
448         {
449
450             return "";
451         }
452
453     }
454 ?>
```