

DIPLOMARBEIT

BackupMagician

Cloudbasiertes Backup-System





Backup-Magician

Diplomarbeit: Cloudbasiertes Backup-System

Projektnummer: ___ BI 17/18

Jahrgang: 6 BBKIF

Schuljahr: 2017/18

Kandidaten: Roland Morhammer

Rene Polak

Richard Rus

Betreuer: Prof. Dipl. Ing. Endre Beda

Prof. Ing. Mag. Manfred Oberkersch



Erklärung

Wir erklären, dass wir die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht haben.

Wien, am 27.06.2018

.....

Roland Morhammer

.....

Rene Polak

.....

Richard Rus



Projektbeschreibung

Unser Projekt, der BackupMagician, ist ein Backupmanagementsystem welches auf folgenden zwei Hauptkomponenten basiert: Dem Server und den Clients. Das Projekt wird demzufolge in einer Server-Client Architektur umgesetzt.

Ziel des Projekts ist es Backups zentral steuern zu können. Dies geschieht über den Server. Der Client fungiert als Zuhörer und Arbeiter.

Es ist sowohl über die auf einem PC installierte Client Software als auch über das Webinterface möglich, einen Benutzer anzulegen, um im System agieren zu können.

Ein Benutzer kann die Berechtigungen für mehrere Clients haben und kann diesen über den Server Aufträge (Tasks) zuweisen, unabhängig davon, ob ein Client gerade on- oder offline ist. Der Server hält die Tasks solange in Evidenz, bis der entsprechende Client online ist und schickt dem Client bei erster Gelegenheit die ihm zugeteilten Tasks.

Der Server „merkt“ sich welche Clients gerade aktiv sind. Zu diesem Zweck authentifizieren sich die Clients beim Server, sobald sie gestartet werden und eine Verbindung zum Server aufbauen können.

Sobald sich ein Client authentifiziert und ein User über den Client eingeloggt hat, fügt der Server dem User die Berechtigung für diesen Client hinzu – die Zuordnung erfolgt über den Benutzernamen und die IP-Adresse des Clients. Von nun an kann der Benutzer dem Client Aufträge erteilen, was entweder lokal über den Client oder über das Webinterface geschieht. In beiden Fällen wird auf ein- und dasselbe Server-Interface zugegriffen.

Sowohl Tasks, die der Server an die Clients schickt, als auch andere



Nachrichten, über die der Server mit den Clients kommuniziert, werden als kompakte JSON-Dateien abgebildet und versandt.

Sobald der Benutzer Aufträge erstellt hat, schickt der Server diese an die entsprechenden, verfügbaren Clients. Die Tasks beinhalten sowohl Daten, die den Umfang des Backups definieren, als auch die für den Ziel-Host (das Cloud-Service) benötigten Zugangsdaten.

Der Client bestätigt dem Server den Empfang der Tasks, und beginnt mit der Verarbeitung der Aufträge. Im Zuge dessen überprüft der Server ob das Backup gerade möglich ist, also ob das Cloud-Service online und genug freier Speicherplatz vorhanden ist. Der Server teilt dem Client das Ergebnis dieser Überprüfung mit und dieser kopiert im Falle der positiven Überprüfung die gemäß des Auftrages zu sichernden Daten auf den Cloud-Speicher.

Nach erfolgreicher Durchführung eines Tasks setzt der Client den Server über den Abschluss des Auftrages in Kenntnis. Ist bei der Durchführung des Tasks clientseitig ein Fehler aufgetreten, so schickt der Client dem Server ein entsprechendes Log.

Um einem aktiven Client einen Auftrag zu erteilen muss sich ein Benutzer nicht unbedingt am entsprechenden Client selbst einloggen. Es besteht die Möglichkeit, in einem beliebigen Browser das Webinterface aufzurufen, sich mit seinem Benutzeraccount einzuloggen und auf diesem Weg den Clients über den Server ihre Tasks zukommen zu lassen.



Project description

Our project, the BackupMagician is a backup management system based on two main components: the server and the clients. The project therefore is realized as a server-client architecture.

The aim of our project is to allow users to control backups from a central point: the server. The client acts as a listener and worker.

A user account can be created by using the client software, which is installed on a PC as well as by using the web interface. After creating a user account the user is able to access the system.

A user can possess rights for multiple clients and assign tasks to them through the server, regardless of the client being online or offline. The server keeps track of existing tasks and sends them to the corresponding client as soon as the client comes online.

The server keeps a list of active clients. When a client is started it connects to the server and authenticates with the server.

As soon as a client has authenticated with the server and a user has logged in to the server through that client the user is granted the right for that client – the user name becomes associated with the IP address of the client. As a result the user can assign tasks to that client using the client locally or using the web interface. The very same server interface is accessed in either case.

Tasks the server sends to a client as well as other messages exchanged by server and clients are represented by and sent as compact JSON files.

Upon creation of tasks by a user the server sends those tasks to the



corresponding available clients. Those tasks contain information about the scope of the backup itself as well as the login information for the targeted cloud service.

Upon receiving a task the client sends a confirmation message to the server and starts processing the task. The server checks if there is enough free space on the cloud service and communicates the result to the client. If there is enough free space on the cloud service the client starts uploading the files to the cloud according to the tasks specifications.

After a client has finished processing a task it informs the server about it. In case there occurred any errors during processing the task the client sends an error log to the server.

It is not necessary to log in through a client to commission a task to that client. It is equally possible to access the web interface from any browser, log in to a user account and commission tasks to clients in this way.



Inhaltsverzeichnis

Projektbeschreibung.....	4
Project description.....	6
Einführung.....	9
Ziele, Nicht-Ziele und Milestones.....	10
System Overview.....	11
System Komponenten Beschreibung.....	11
Software Komponenten.....	11
Der BackupMagician Core.....	11
BM Core - Erhebung der Daten & Verarbeitung der Daten.....	12
BM Core - Vorbereitung der Daten für die Sicherung.....	13
BM Core - Durchführung der Sicherung.....	14
Der BackupMagician Server.....	15
Der Kommunikator.....	17
System Design.....	18
Technologie Auswahl.....	18
Systemarchitektur Flowchart	20
Aufbau des BackupMagician.....	23
System Implementierung.....	24
Entwicklung des Kommunikators.....	24
Client Software.....	25
BackupMagician Setup.....	25
Aufbau des BackupMagician Setup.....	25
Server Software.....	32
Web Monitoring Software.....	33
Offizielle Website des BackupMagician.....	33
Screenshots der Website.....	36
Erstellung des Anmeldungs- und Registrierungs-Formulars.....	39
Erstellung einer Datenbank.....	44
Verknüpfung des Formulars mit der Testdatenbank.....	49
Design des Logos.....	53
Design der Icons.....	55
Arbeitsaufteilung.....	56



Einführung

Wir speichern heutzutage einen großen Teil unserer Daten auf Cloud Services, vor allem Cloud-Speicher wie Google Drive, Dropbox oder Icloud und nutzen diese auch um unsere Daten miteinander zu teilen.

Cloud Services nehmen einen immer wichtigeren Platz in unserem Leben ein; Aber gibt es eine Software, mit Hilfe derer wir diese verwalten und unseren Alltag vereinfachen können, vom einfachen Anwender bis hin zum großen Unternehmen?

Bisher gab es das noch nicht - aber jetzt schon!

Unser Projekt ist genau das: Eine einfach zu bedienende Software, welche als Managementsystem für ebendiese Cloud Speicher dient. Mit wenigen Klicks werden Sicherungen verwaltet und gesteuert, von Otto-normal-Verbraucher wie von großen Unternehmen.

Der Name dieser Software ist „BackupMagician“ und wenn man nicht wüsste was passiert – man würde glatt denken es wird gezaubert!

Um diesen Trick zu bewältigen nutzen wir eine ausgeklügelte Kombination aus kleinen Programmen und verschiedenen Technologien, welche es dem Anwender ermöglichen dieses Verwaltungsaufwandes mit wenigen, einfachen Klicks Herr zu werden.



Ziele, Nicht-Ziele und Milestones

Projektziele

1. Erstellung eines funktionalen Prototypen mit allen wichtigen Funktionen
2. Erstellung der Client-Backupsoftware
3. Erstellung des Webinterface
4. Erstellung der Serververwaltungssoftware
5. Realisierung der Kommunikation zwischen Server und Client
6. Umsetzung eines Mechanismus zum Senden von Files an Cloud Services

Nicht-Ziele:

1. Usermanagmentsystem für Authentifizierung (oder z.b OAuth)
2. Super-ausgereifte Networksecurity
3. Extrem belastbare Server-Software welche Millionen von Nutzern bewältigt.

Projekt-Zeitplan

	Sep.	Okt.	Nov.	Dez.	Jan.	Feb.	März	April	Mai	Juni
Planung des Projektes										
Software										
Testen der Software										
Dokumentation										



System Overview

System Komponenten Beschreibung

Software Komponenten

Der BackupMagician Core

Das BackupMagician Core Programm ist ein integraler Hauptbestandteil der Software Struktur. Durch ihn werden die Backups durchgeführt und abgeschlossen.

Der Flowchart zeigt anschaulich den Aufbau des Programms, um die Funktionsweise aber wirklich gut verstehen zu können folgen nun einige Anmerkungen:

Das Programm ist grob in folgende Abschnitte gegliedert:

- Auslesen der Daten aus dem als Übergabeparameter erhaltenen Task-JSON (wird von BackupMagician Setup an einem fixen ["hardgecodeten"] Ort gespeichert)
- Verarbeitung dieser Daten und Erstellung der Ordner-Objekt Struktur
- Plattformspezifische Vorbereitung der Daten für die Sicherung
- Durchführung der Sicherung und Beendigung des Programms

Im Hauptprogramm (der Funktion "main") des BackupMagician Core Programms wird das Hauptstück, der QbackupManager erstellt. Diese Klasse wird in einem Pointer-Objekt gespeichert.

Es wird die Funktion "Init" aufgerufen, welche die Erhebung und Verarbeitung der Daten durchführt. Mehr dazu im Punkt "Erhebung der Daten".



Backup-Magician

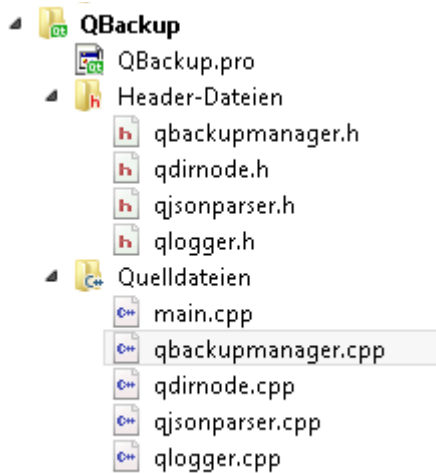


Abbildung 1: Die Klassen des BackupMagician Core Programms

Abbildung 1 zeigt die wichtigsten Klassen des BackupMagician Core Programms.

Die Klasse QbackupManager ist für die Steuerung des Backups zuständig.

Die Klassen QdirNode und QjsonParser sind für die Verarbeitung und Datenerhebung zuständig.

Die Klasse Qlogger erstellt im Verzeichnis /log eine Log-Datei und ermöglicht es dem Benutzer eventuell auftretende Fehler zu erkennen und zu beheben.

BM Core - Erhebung der Daten & Verarbeitung der Daten

Im ersten Schritt wird hier ein QjsonParser Objekt erstellt. Dieses liest aus einem im BackupMagician Core vordefinierten Ordner (/config) die JSON-Datei aus.

Die ausgelesenen Informationen werden daraufhin in dem QjsonParser Objekt gespeichert und sind über "Get"-Funktionen abrufbar.

Anhand dieser erhobenen Daten wird das QdirNode Objekt erstellt. Dieses



bildet dann die eigentliche Ordner Struktur anhand der Angabe ab.

Die Dateien werden auch gespeichert.

Dies wurde mittels eines Rekursiven Aufrufs des Konstruktors umgesetzt.

Der Konstruktor filtert die ins Suchschema passenden Ordner heraus und erstellt entsprechende Unterordner. Dies geschieht rekursiv so lange, bis der Konstruktor ans Ende einer Unterordner-Kette gelangt ist, also kein weiter Unterordner mehr da ist, der übergeben werden könnte.

Als abschließender Schritt wird die erstellte Struktur einer Variable zugeordnet. Damit ist der erste große Teil des Backup-Prozesses durchgeführt: Die Erhebung und Verarbeitung der Daten.

BM Core - Vorbereitung der Daten für die Sicherung

Sobald die Funktion Init durchlaufen wurde, wird die nächste Funktion im Hauptprogramm aufgerufen: "initBackup".

Diese Funktion hat die Aufgabe, anhand der ihr übergebenen Parameter, die zugehörigen Funktionen für die Sicherung selbst aufzurufen.

Die Funktion überprüft welcher "SaveType" vorhanden ist (also welches Cloud Service für das Backup benutzt werden soll) und ruft dann die dementsprechende Unterfunktion auf. Zur Zeit ist nur Dropbox implementiert.

Die Funktion liest dann alle notwendigen Daten aus, und erstellt in Vorbereitung für den nächsten Schritt eine Datei "temp.txt".

Mit dieser Datei als Parameter wird die Information, welche Ordner gesichert werden sollen, an ein Python Skript übergeben.

Eine Kommunikation mit Dropbox ist mittels QT nur schwer realisierbar,



deshalb wurde dazu die von Dropbox bereits geschriebene SDK Python Library verwendet.

Sobald alle Daten vorbereitet wurden ruft das BackupMagician Core Programm einen Qprocess auf, welcher das Python Skript mit bestimmten Parametern aufruft. Mit vollendetem Ablauf des Python Skripts ist auch der Backupvorgang abgeschlossen und das BackupMagician Core Programm beendet sich.

BM Core - Durchführung der Sicherung

Wie im letzten Punkt bereits erwähnt wurde, wird der Upload der Daten auf Dropbox durch ein Python Skript abgewickelt.

Das Python Skript bekommt alle für die Sicherung relevanten Informationen als Parameter übergeben.

Das Python Skript erledigt in seinem Ablauf folgende Aufgaben:

- Anpassung der Parameter für Dropbox
- Prüfung, ob genug Speicherplatz auf Dropbox vorhanden ist
- Hochladen der zu sichernden Dateien auf Dropbox
- Rückmeldung per Rückgabewert an das BackupMagician Core Programm bei Skriptende

Nach Erhalt der Rückmeldung vom Python Script gibt das BackupMagician Core Programm seinerseits eine Rückmeldung an das BackupMagician Setup zurück. Das BackupMagician Setup schreibt die Rückmeldung in seinen QTextBrowser und setzt den Benutzer so über die erfolgreiche Verarbeitung des Tasks in Kenntnis.



Der BackupMagician Server

Der BackupMagician Server ist das Hauptstück des Projekts. Er nimmt Pakete in Empfang, die ihm per TCP zugeschickt werden und verarbeitet diese. Diese Pakete enthalten JSON-Dateien mit wichtigen Inhalten.

Clients können vier verschiedene Anfragen (Abbildung 2) stellen:

```
void MyThread::readyRead()
{
    QByteArray Data = socket->readAll();
    qDebug() << socketDescriptor << "Data in: " << Data;

    QJsonDocument jsondoc = QJsonDocument::fromJson(Data);
    QJsonObject jsonobj = jsondoc.object();
    QString type = jsonobj["type"].toString();

    if (!type.isEmpty())
    {
        QString user = jsonobj["username"].toString();
        QString pass = jsonobj["password"].toString();

        if(type == "task") {

        }

        if(type == "dirs")
        {
            if(db.userLogin(user,pass)) { ... }
            else { ... }
        }

        if(type == "clouddata") { ... }

        if (type == "authentication") { ... }

        if (type == "register")
        {
            if(!db.checkUser(user)) { ... }
            else { ... }
        }

    }

    else socket->write("Warning: Received JSon without \"type\" key");
}
```

Abbildung 2: Die readyRead-Funktion der MyThread Klasse des Servers



- Authentifizierung
- Registrierung
- Ordner
- Cloud Service Zugangsdaten

Zusätzlich gibt es noch zur Verarbeitung der Aufgaben eigene Task-Dateien, welche der Server an den jeweils entsprechenden Client weiterschickt. Jedes dieser Pakete enthält immer den Benutzernamen mit zugehörigem Passwort, damit der Server sie richtig zuordnen kann.

Auf dem Server liegt ein CSV Speicher. Der Server liest diesen aus und vergleicht den ausgelesenen Benutzernamen und das ausgelesene Passwort mit dem im erhaltenen Paket enthaltenem Benutzernamen und Passwort. Ist das Ergebnis eine Übereinstimmung, so wird das Paket weiter verarbeitet.

Es war ursprünglich geplant, eine MySQL Datenbank an den Server anzubinden, jedoch bietet QT selbst keine Klassen zur Kommunikation mit einer MySQL-Datenbank und die Anbindung einer MySQL-Datenbank wäre aufwändiger und komplizierter gewesen als die Lösung, die zu verwaltenden Informationen in CSV-Dateien zu schreiben und aus diesen wieder auszulesen.

Der Punkt "Cloud Service Zugangsdaten" konnte aufgrund Zeitmangels leider nicht mehr realisiert werden.

Tasks (inklusive Schlüssel für Dropbox) werden im Webinterface erstellt und an den Server geschickt. Der Server sendet die Tasks weiter an das BackupMagician Setup Programm und dieses wiederum übergibt den Task



an das BackupMagician Core Programm, welches dann mit Hilfe des Python Skripts den Upload der Ordner und Dateien durchführt. Der Server steht in dieser Architektur im Zentrum der Vorgänge und steuert diese.

Der Kommunikator

Der Kommunikator umfasst sowohl die Teile der Client Software, die dem Datenaustausch zwischen Client und Server dienen, als auch die entsprechenden Teile der Server Software. Die mitzuteilenden Informationen werden dabei in JSON-Dokumente verpackt und per TCP Protokoll vom Client zum Server oder umgekehrt übertragen. Um eingehende Verbindungsanfragen zu verarbeiten wird sowohl am Client als auch am Server ein Objekt der QTcpServer-Klasse verwendet (passiver Teil des Verbindungsaufbaus). Dem aktiven Verbindungsaufbau, also dem initiieren einer Verbindung dient sowohl server- als auch clientseitig ein Objekt der Klasse QTcpSocket.

Hinsichtlich des Kommunikators besteht der wesentliche Unterschied zwischen Server und Client naturgemäß darin, dass ein Client mit nur einem Server in Verbindung steht, während der Server Anfragen von mehreren Clients zur gleichen Zeit verarbeiten können muss. Um dies zu ermöglichen besitzt der Server für jeden Client einen eigenen Thread, sprich je aktivem Client ein Objekt der „MyThread“ Klasse, die die Funktionalität der QThread-Klasse geerbt hat. Zusätzlich ist die „MyThread“ Klasse mit einem QTcpSocket ausgestattet. Registriert das Objekt der „MyServer“ Klasse (welche das QTcpServer-Objekt enthält) eine eingehende Clientverbindung, so erzeugt es ein Objekt der „MyThread“ Klasse und gibt die Verbindung an dieses weiter. Auf diese Weise entsteht für jeden Client, der sich verbindet ein eigener Thread, der die per TCP eingehenden JSON-Dokumente genau dieses Clients entgegennimmt.



Im Gegensatz zur Verarbeitung von empfangenen Nachrichten (JSON-Dokumenten) unterscheiden sich Server und Client beim Verschicken von Nachrichten nicht prinzipiell. So bauen sowohl Server als auch Client dazu nur vorübergehend eine Verbindung mit Hilfe des QTcpSocket-Objekts auf, die lediglich der einmaligen Übermittlung eines einzelnen JSon-Dokuments dient.

System Design

Technologie Auswahl

Welche Technologien wurden benutzt?

- - QT C++
- - Python
- - JSON
- - HTML5
- - CSS 3.0
- - JavaScript + JQuery
- - TCP/IP + Webserver
- - PHP

Weshalb wurden genau diese benutzt?

Es gibt verschiedene Gründe weshalb wir gerade diese Technologien benutzt haben; Der naheliegendste Grund ist, dass wir mit diesen unser Projekt realisieren konnten.

In vielen Fällen lag es in der Lizenzierung begründet. Diese Technologien boten die Möglichkeit, unser Produkt ohne großen Aufwand zu



veröffentlichen.

Der wichtigste Grund aber war wahrscheinlich vor allem, dass wir mit diesen Technologien bereits in der Schulzeit gearbeitet und somit schon zuvor Erfahrung gesammelt hatten.

Jedes Projektmitglied hatte bereits Erfahrung mit mindestens 3-4 dieser Technologien und die Kenntnisse, die noch fehlten könnte man sich im Zuge der Arbeit am Projekt noch erarbeiten.

Die Client- und Serversoftware wurde in QT C++ geschrieben; der Server nutzt außerdem noch JSON und der Client Python und JSON. Die Website wurde durch Nutzung der dafür typischen Technologien erstellt: HTML, CSS und JavaScript/Jquery.



Systemarchitektur Flowchart BackupMagician Setup

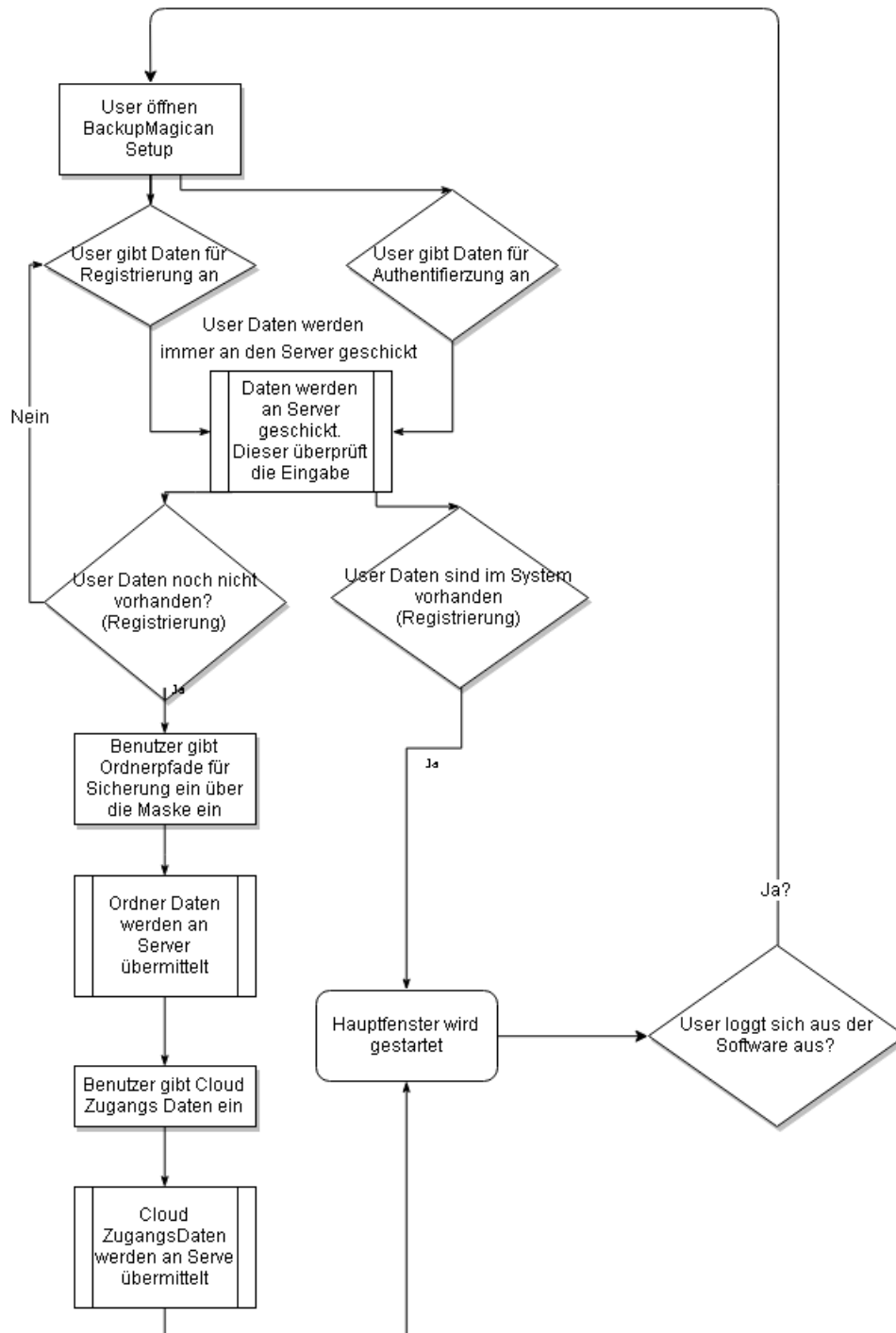


Abbildung 3: Flowchart des BackupMagician Setup



LEGENDE:

- Normale rechteckige Kästchen sind Vorgänge, welche der User steuert.
- Rechteckigen Kästchen mit Extra-Unterteilungen sind Aufgaben, welche Der Server in diesem Zusammenhang erledigt.
- Deltoide sind Abschnitte im Prozess, welche einer Überprüfung bedürfen und im Normalfall zwei Ergebnisse haben und so den Prozess weiterführen oder diesen wiederholen lassen.

Anmerkung:

Das Flowchart ist abgeschlossen sobald der Benutzer (wieder) ins Hauptfenster gelangt.

Hier hat er zusätzlich noch die Möglichkeit sich wieder auszuloggen. Entscheidet er sich dafür wird er wieder auf den ersten Bildschirm zurück gesetzt: Den Bildschirm zur Anmeldung bzw. Registrierung.

Im Flowchart wurde nicht die Möglichkeit zur Editierung der Eingaben abgebildet. Der Benutzer hat in jedem Abschnitt die Möglichkeit, Daten hinzuzufügen (zu sichernde Ordner oder Cloud-Zugangsdaten) oder diese wieder zu löschen.

Eine verbale Beschreibung des BackupMagician Setup ist im Abschnitt "BackupMagician Setup" bzw. "Backupmagician Setup Aufbau" dieser Diplomarbeit zu finden.



Flowchart BackupMagician Core

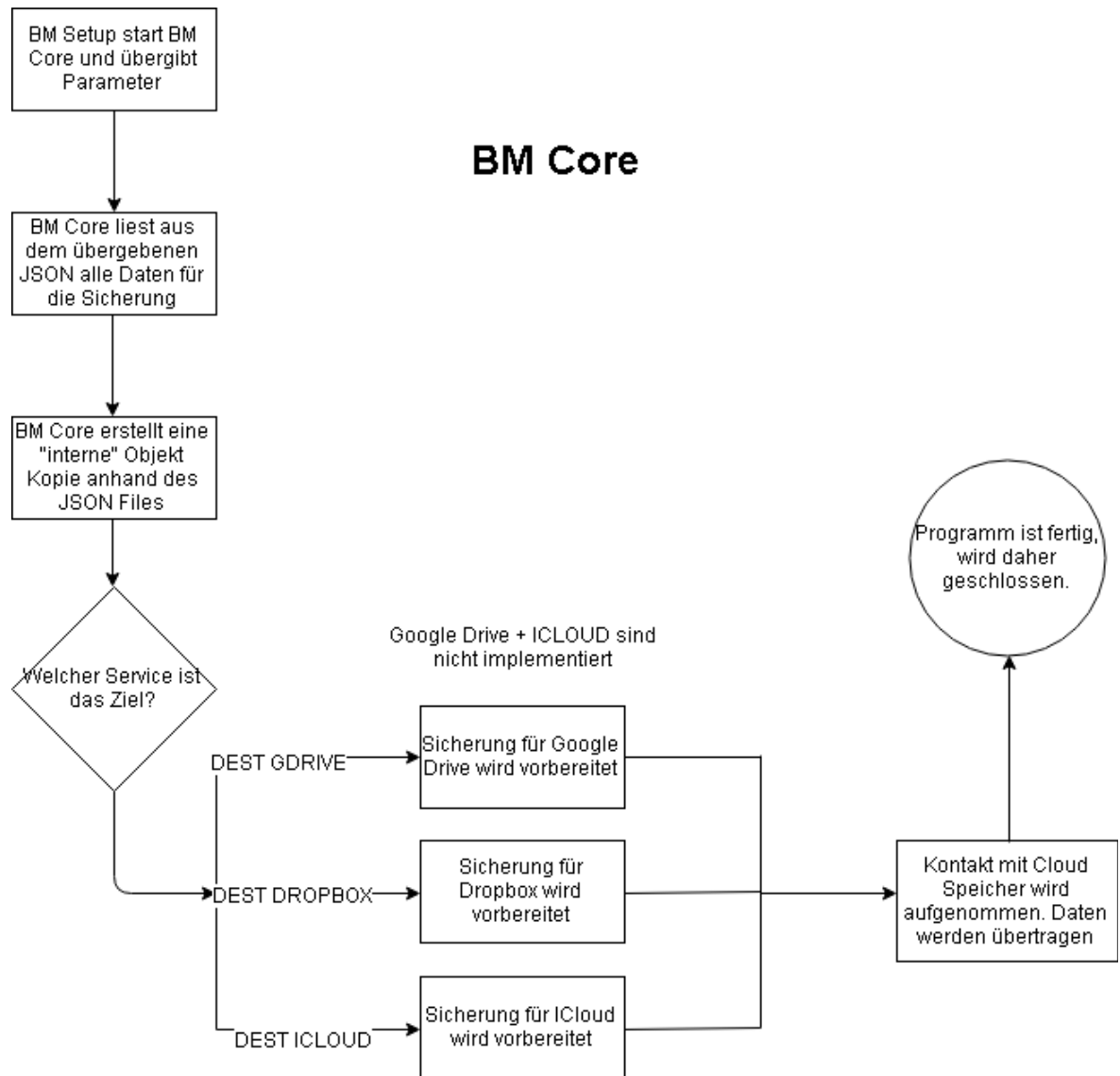


Abbildung 4: Flowchart des BackupMagician Core Programms

LEGENDE:

- Rechteckige Kästchen repräsentieren Programmabschnitte
- Deltoide repräsentieren Entscheidungen
- Der Kreis symbolisiert, dass sich das BackupMagician Core Programm nach dem erfolgreichen Durchlauf schließt, den sogenannten Endpunkt.



Anmerkung:

Das BackupMagician Core Programm wird pro Task einmal aufgerufen. Das bedeutet, dass sich dieser Vorgang wiederholen kann; Er wird nicht dauerhaft ausgeführt.

Aufbau des BackupMagician

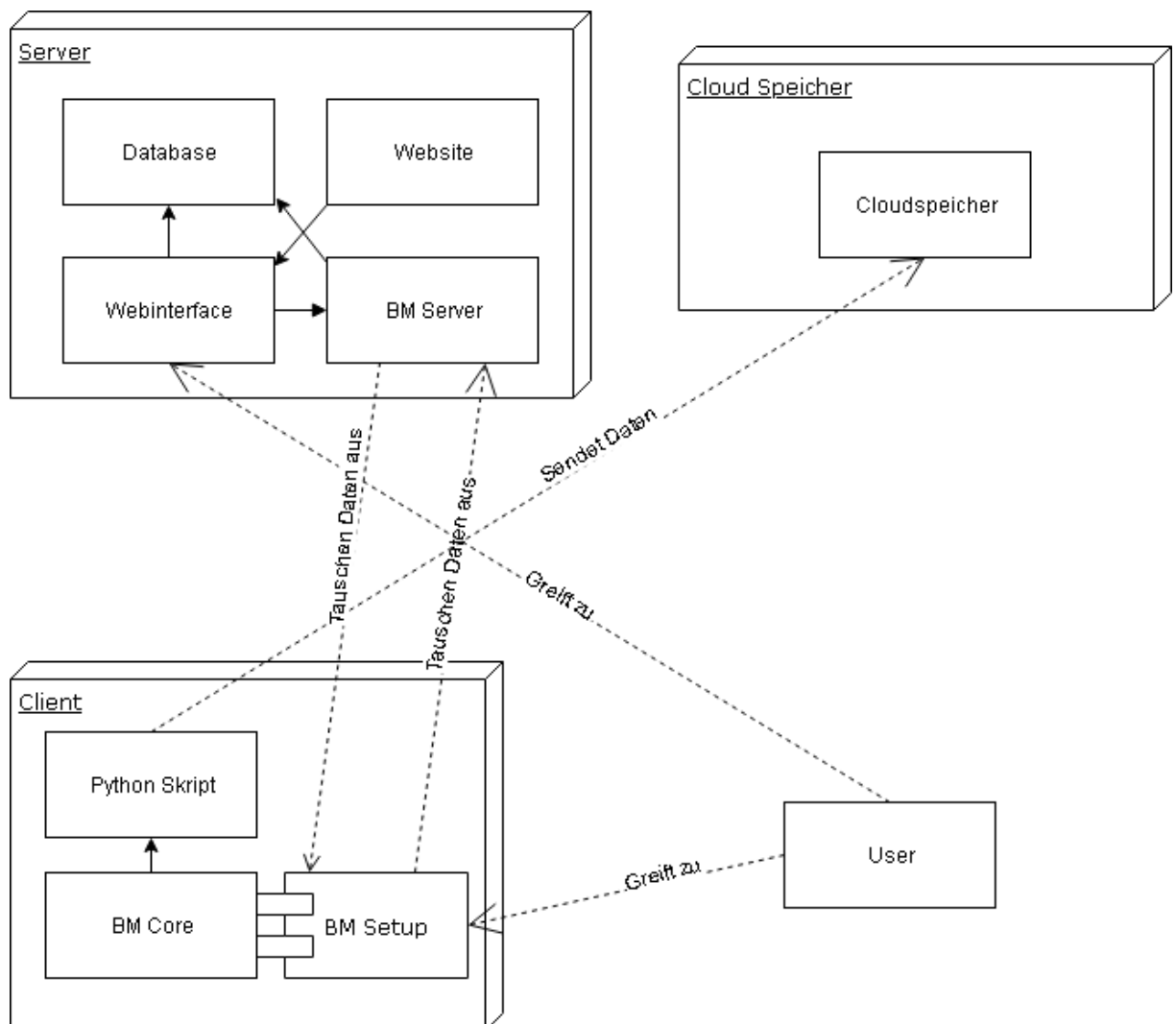


Abbildung 5: Aufbau des BackupMagician

LEGENDE:

- Viereckige Blöcke repräsentieren eigene Software Komponenten. Die



Ausnahme stellt hier der Benutzer ("User") dar, dieser ist eine physische Person, welche das System nutzt.

- Gestrichelte Pfeile repräsentieren Kommunikationswege.
- Durchgängige Pfeile repräsentieren Schnittstellen bzw. direkte Anbindungen der Softwarekomponenten an andere Softwarekomponenten.

System Implementierung

Entwicklung des Kommunikators

Da die Entwicklung des Kommunikators in einem Stadium des Projekts begann, als noch kein Prototyp der Client- oder Serversoftware vorhanden war, mussten im Rahmen der Entwicklung der Kommunikationsroutinen ein Client- und ein Serverdummy programmiert werden. Zuerst wurde sowohl am Client- als auch am Serverdummy eine externe Bibliothek eingebunden: der „QtWebApp HTTP Webserver“ von www.stefanfrings.de. Die Idee war, dass sowohl Server als auch Client als HTTP Server auf eingehende, als HTTP Request verpackte, JSON-Dokumente lauschen würden. Es stellte sich in der Arbeit mit der „QtWebApp HTTP Server“-Bibliothek jedoch heraus, dass ein Weiterleiten von Informationen aus der Klasse, die die empfangenen HTTP-Requests auswertete, unmöglich schien, da der Implementierungsversuch einer Signal-Slot-Verbindung aus der Klasse heraus scheiterte. Obwohl die connect()-Funktion, die das Signal mit dem Slot verbinden sollte TRUE zurücklieferte (sprich: rückmeldete, dass Signal und Slot erfolgreich verbunden wurden) führte ein in gerade erwähnter Klasse emittiertes Signal niemals dazu, dass der entsprechende Slot aufgerufen wurde. Die Nutzung dieser externen Bibliothek von www.stefanfrings.de stellte sich also als Sackgasse heraus



und es ergab sich die Notwendigkeit, die Programmroutinen, die es dem Client bzw. Server ermöglichen, Daten vom jeweils anderen zu empfangen, selbst zu schreiben. Welche Klassen dazu wie genutzt wurden ist im Abschnitt "Der Kommunikator" dieser Diplomarbeit ersichtlich.

Client Software

BackupMagician Setup

Eines der Programme, die es dem Anwender ermöglichen den Backupmagician zu nutzen ist das BackupMagician Setup. Dieses bietet mit Hilfe der Funktionen des Kommunikators die Möglichkeit, folgende Aufgaben abzuwickeln:

1. Registrierung eines Benutzers
2. Authentifizierung eines Benutzers
3. Erstmalige Definition der Ordner, welche für Backups zugänglich sein sollen
4. Hinzufügen der Zugangsdaten für das Cloud Service

Aufbau des BackupMagician Setup

Der Client ist in 4 wichtige Abschnitte unterteilt:

1. Anmeldungs- und Registrierungsformular
2. Im Fall der Registrierung: Auswahl der Ordner, die gesichert werden sollen
3. Überprüfung der Benutzerdaten und Eingabe der Cloud Service Zugangsdaten
4. Hauptbildschirm



Dies sind die 4 wichtigsten Elemente, um die Kommunikation und den Ablauf der Sicherungsroutine zu ermöglichen. Auf der Website wird noch detaillierter beschrieben, wie die einzelnen Cloud Services hinzugefügt werden können (derzeit noch nicht vorhanden).

1. Anmeldung und Registrierung

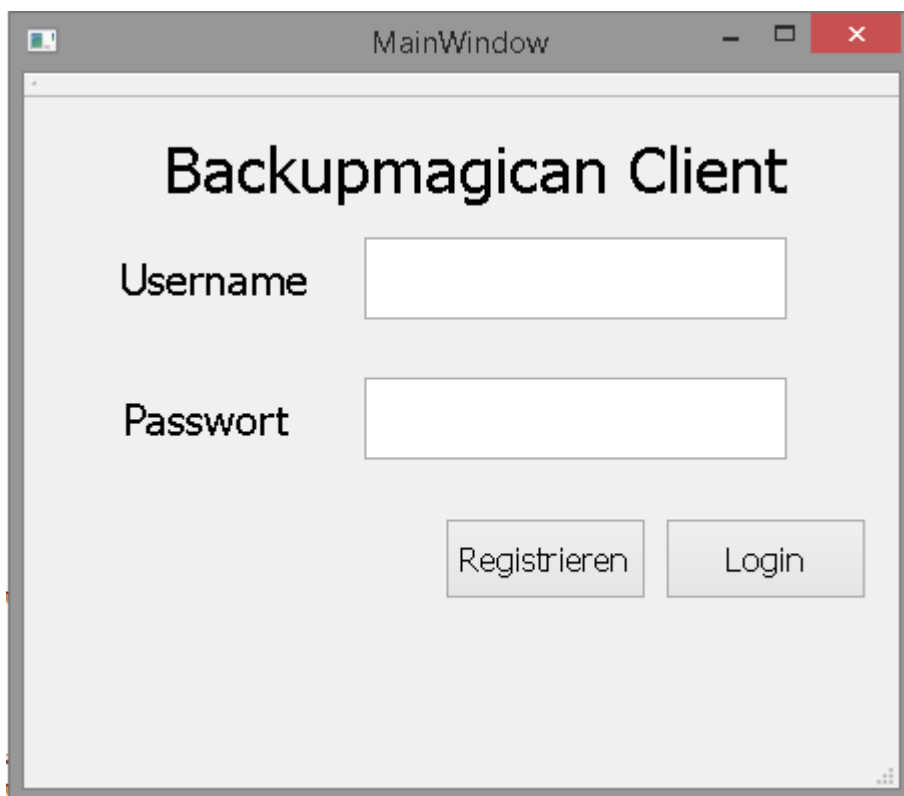


Abbildung 6: BackupMagician Setup - Anmeldung und Registrierung

Hier hat der Benutzer die Möglichkeit ein Benutzerkonto anzulegen oder sich mit seinem vorhandenen Konto anzumelden.

Versucht der Benutzer ein Konto mittels des „Registrieren“-Buttons anzulegen, schickt der Client ein Paket an den Server, um zu prüfen ob der Benutzername bereits vergeben ist. Falls der Server rückmeldet, dass der



Benutzername bereits vergeben ist, kann der Benutzer einen neuen Benutzernamen eingeben.

Sobald sich der Benutzer mit korrektem Benutzernamen und Passwort anmeldet erhält er vom Server eine Bestätigung über die erfolgreiche Anmeldung und gelangt unmittelbar zum Hauptbildschirm. Sofern der Server für den Client bereits eine anstehende Aufgabe (Backup Task) hat, wird diese dem Client sofort nach der Anmeldung zugestellt.

Ein Benutzer kann nur Clients Aufgaben zuteilen und die Aufgaben können von den Clients nur dann durchgeführt werden, wenn er auf diesen angemeldet ist.

2. Setup Ordnerauswahl



Abbildung 7: BackupMagician Setup - Ordnerauswahl



Das "Setup" Fenster bietet dem Benutzer die Möglichkeit auszuwählen, welche Ordner gesichert werden sollen.

Klickt der Benutzer auf „Ordner Hinzufügen“ erscheint ein Fenster, das den Benutzer die Verzeichnisstruktur seines Computers navigieren und einen Ordner auswählen lässt.

Hat der Benutzer die Auswahl der zu sichernden Ordner abgeschlossen, kann er dies mit einem Klick auf „Fertig“ bestätigen oder mit „Zurück“ die Ordnerauswahl abbrechen und zur Anmeldung und Registrierung zurückkehren.

Hat der Benutzer die Auswahl der zu sichernden Ordner abgeschlossen und mit „Fertig“ bestätigt, so schickt der Client die Liste der ausgewählten Ordner an den Server. Dieser speichert diese Liste für sich.



3. Setup Cloudspeicher Zugangsdaten

Fast geschafft

Überprüfen sie bitte ihre Eingaben!

Zusätzlich haben sie jetzt die Möglichkeit ihre Cloud Speicher Zugangsda

Username:

Passwort:

[Empty text box]

Fertig Zurück Cloud Daten hinzufügen

Abbildung 8: BackupMagician Setup - Cloudspeicher Zugangsdaten

Im zweiten Setup-Fenster kann der Benutzer seinen Benutzernamen und sein Passwort überprüfen und falls er diese ändern möchte, so kann er auf „Zurück“ klicken, um über das erste Setup-Fenster durch einen weiteren Klick auf „Zurück“ wieder zur Anmeldung und Registrierung zu gelangen.

Durch einen Klick auf „Cloud Daten hinzufügen“ öffnet sich ein Fenster, in dem der Benutzer seine Zugangsdaten für Dropbox, Google Drive und so weiter eingeben kann. Diese werden dann diesem Benutzer zugeordnet und werden vom Server auch genutzt, wenn er über das Webinterface angesteuert wird.

Diesen Schritt kann der Benutzer überspringen, da er die Eingabe seiner



Cloud-Zugangsdaten auch über die Website abwickeln kann.

4. Hauptbildschirm

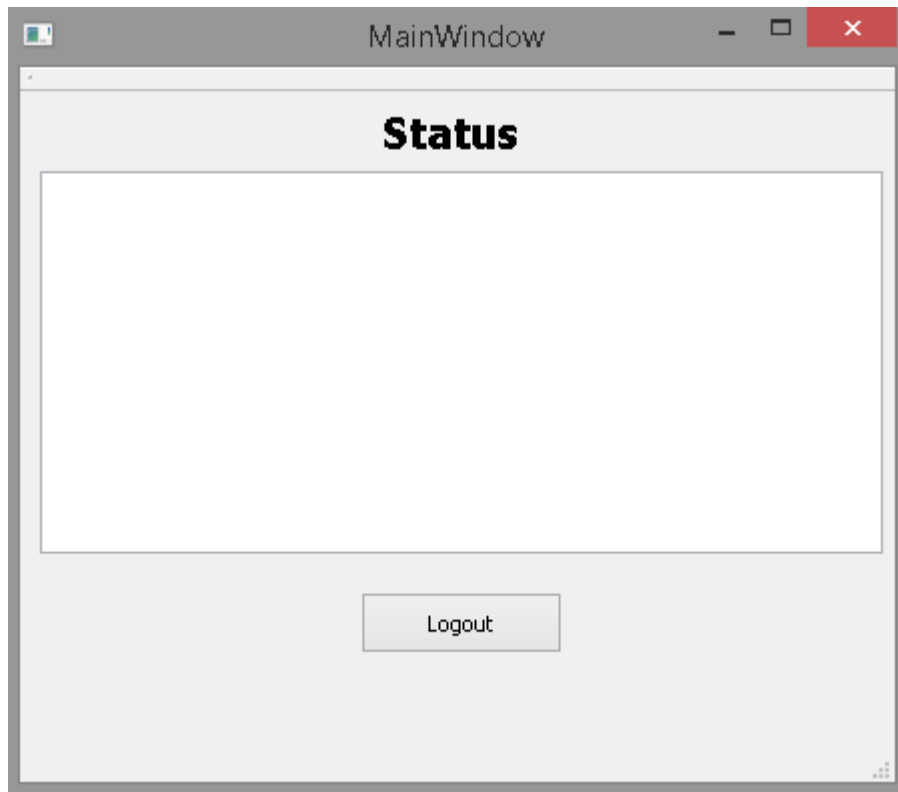


Abbildung 9: BackupMagician Setup - Hauptbildschirm

Sobald der Benutzer die Anmeldung und das Setup abgeschlossen hat, gelangt er auf den Hauptbildschirm. Auf diesem hat der Benutzer die Möglichkeit, die Aktivitäten der Software mitzuverfolgen: es ist ihm ersichtlich, welche Aufgabe der Client gerade durchführt, selbst wenn diese Aufgabe über das Webinterface in Auftrag gegeben wurde.

Hat der Client sämtliche Aufgaben abgeschlossen, so kann der Benutzer den Client durch einen Klick auf „Logout“ bequem verlassen, was zur Folge hat, dass der Server über diesen Client keine Backup-Tasks für den ausgeloggten Benutzer mehr durchführen kann.



BackupMagician Core

Das Kernstück des BackupMagician ist das Programm, das den eigentlichen Upload auf das Cloudservice leistet.

Dieses ist dem Client zugehörig und hat folgende Aufgaben:

1. Aufgaben verarbeiten
2. Ordner bzw. Ordnerstruktur anlegen
3. Durchführung des Uploads

1. Task verarbeiten

Das BackupMagician Core Programm bekommt Aufgaben in JSON-Notation vom BackupMagician Setup, welches die Aufgaben, die der Server an ihn schickt an das BackupMagician Core Programm weiterleitet.

Das BackupMagician Setup Programm speichert die vom Server in JSON-Notation erhaltenen Aufgaben physisch als JSON-Dateien und ruft daraufhin das BackupMagician Core Programm (bmcore.exe) mit der JSON-Datei als Parameter auf. Wenn das BackupMagician Core Programm startet, liest es die Informationen aus der JSON-Datei aus und speichert sie zur Verarbeitung zwischen.

Dabei ist zu beachten, dass das BackupMagician Core Programm mit nur einer JSON-Datei als Parameter aufgerufen wird, nicht mit mehreren auf einmal.

2. Ordner- und Dateistruktur anlegen

Als nächsten Schritt bildet das BackupMagician Core Programm die relevante Ordner- und Dateistruktur ab. Um das zu realisieren wird die QT Creator Klasse „QDir“ genutzt, welche es ermöglicht, die Ordner und



Dateien rekursiv auszulesen, um so intern eine neue Struktur aufzubauen, die nur aus relevanten Ordnern besteht, nämlich solchen, die Dateien enthalten, deren Dateinamen-Suffix in das Suchschema passt.

Ist dieser Schritt beendet, kann ein Backup erstellt werden, indem die Dateipfade durch selbstgeschriebene Funktionen abgefragt werden.

3. Aufgabe verarbeiten

Der letzte Schritt ist einfach:

Das BackupMagician Core Programm hat sich die zur Durchführung der Aufgabe notwendigen Informationen aus der als Kommandozeilenparameter übergebenen JSON-Datei intern zwischengespeichert und weiß nun, welche Ordner und welche Dateien wohin gespeichert werden sollen. Als nächstes erzeugt es eine Datei, welche eine Liste der hochzuladenden Dateien enthält und übergibt diese Datei, gemeinsam mit den weiteren, für den Upload nötigen Informationen, als Kommandozeilenparameter an ein Python Skript. Dieses Python Skript kommuniziert dann direkt mit Dropbox und lädt die zu sichernden Dateien, Datei für Datei, hoch.

Der Umweg über das Python Skript wurde gewählt, da keine QT Libraries existieren, welche eine Kommunikation mit Dropbox unterstützen und eine Realisierung in QT Creator zu umständlich wäre.

Server Software

Näheres zur Funktionsweise des Servers ist im Abschnitt "BackupMagician Server" im Kapitel "Systemkomponenten Beschreibung" erläutert.



Web Monitoring Software

Anmerkung: Es hätte auf Anfrage von Herrn Professor Beda möglich sein sollen, den Ablauf einer Aufgabe in Echtzeit zu verfolgen: von der Erstellung des Tasks, über den Empfang des Tasks durch den Client, einer Rückmeldung, sobald der Task vom Client bearbeitet wird, bis hin zu einer Rückmeldung, sobald die Bearbeitung des Task fertiggestellt wurde.

Diese Funktionalität konnte aufgrund Zeitmangels leider nicht mehr implementiert werden.

Offizielle Website des BackupMagician

Kaum etwas eignet sich zur Präsentation und Anpreisung einer Software besser als eine Website.

Die Website ist an existierende populäre Websites für Software angelehnt.

Hauptaugenmerk liegt hierbei auf Einfachheit.

Für die Website wurden natürlich HTML, CSS und Javascript/Jquery genutzt, es gibt aber auch eine MySQL Anbindung.

Die Website besteht aus 4 Seiten:

- Home
- Technische Details
- Download
- Login

Jede dieser Seiten erfüllt dabei ihren eigenen Zweck, die Loginseite ist aber besonders da sie das Bindeglied von Website zu Webinterface bzw. zur Beschaffung der nötigen Software für die Nutzung von BackupMagician



darstellt.

Die Website selbst wurde mit einem Template erstellt, welches im Footer referenziert wurde. Auch Impressum und AGBs sind vorhanden; Eine Cookie-Bemerkung zwecks DSGVO wurde noch nicht hinzugefügt.

Erstellt wurde die Website von Projektmitarbeiter Polak Rene mit Hilfe von Rus Alex als Tester.

Home

Die Home Seite hat die einfache Aufgabe dem potentiellen Benutzer die Software näher zu bringen: einfache Grafiken, ein überzeugender Slogan und das Ganze einfach aufbereitet.

Der Betrachter weiß nach kurzem Durchlesen, was er bekommt und was er damit tun kann.

Technische Details

Die Technischen Details sind für alle „Nerds“ unter unseren Besuchern.

Dem interessierten Besucher der Website wird hier etwas genauer beschrieben, was die Software im Detail macht.

Es werden natürlich nicht alle Informationen Preis gegeben, aber dem Besucher wird vermittelt was wie abläuft und was genau die Software macht.

Download

Die Download Seite ist an Githubs Downloadseite angelehnt.

Sollte das Projekt kommerziell genutzt werden hat der Besucher der Website hier die Auswahl zwischen mehreren Setups. Derzeit ist nur eine 64 Bit Version vorgesehen.



Backup-Magician

Login

Die Loginseite ist wie bereits weiter oben erwähnt das Bindeglied zwischen Webinterface und Website bzw. Benutzer und Software.

Der Benutzer erhält hier die Möglichkeit sich anzumelden, um daraufhin mittels des Webinterfaces Tasks zu erstellen, Werte zu überprüfen oder sich neu zu registrieren. Hier erhält der Benutzer Zugang zum Webinterface.



Backup-Magician

Screenshots der Website



Abbildung 10: Screenshot der Website (1)

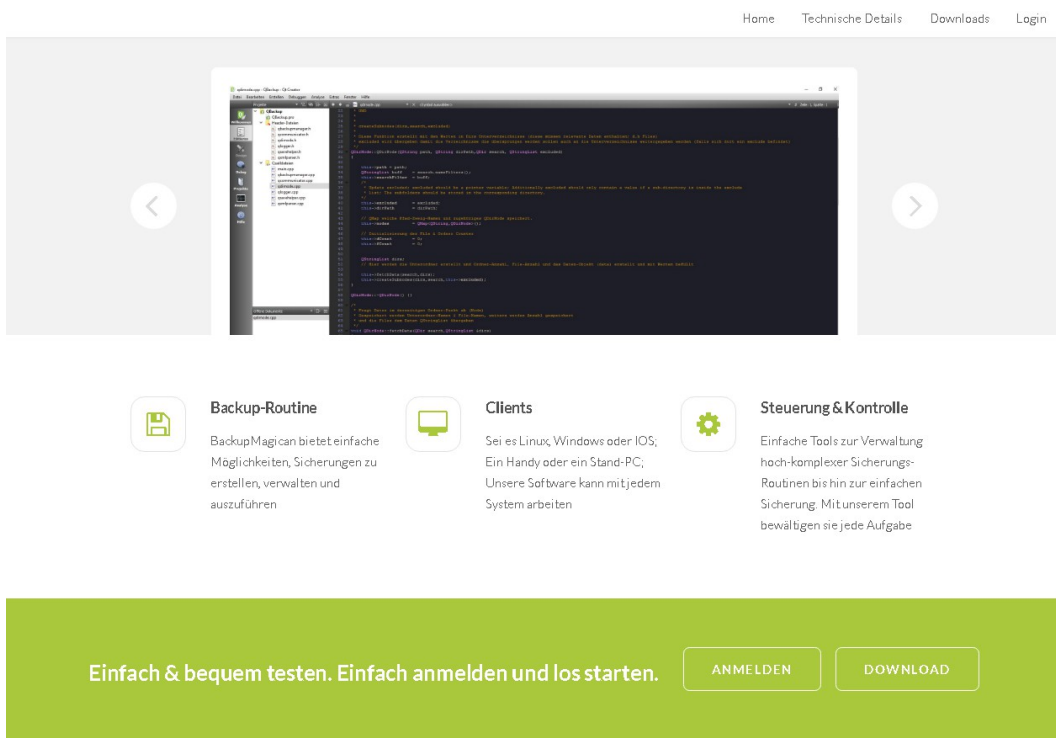


Abbildung 11: Screenshot der Website (2)

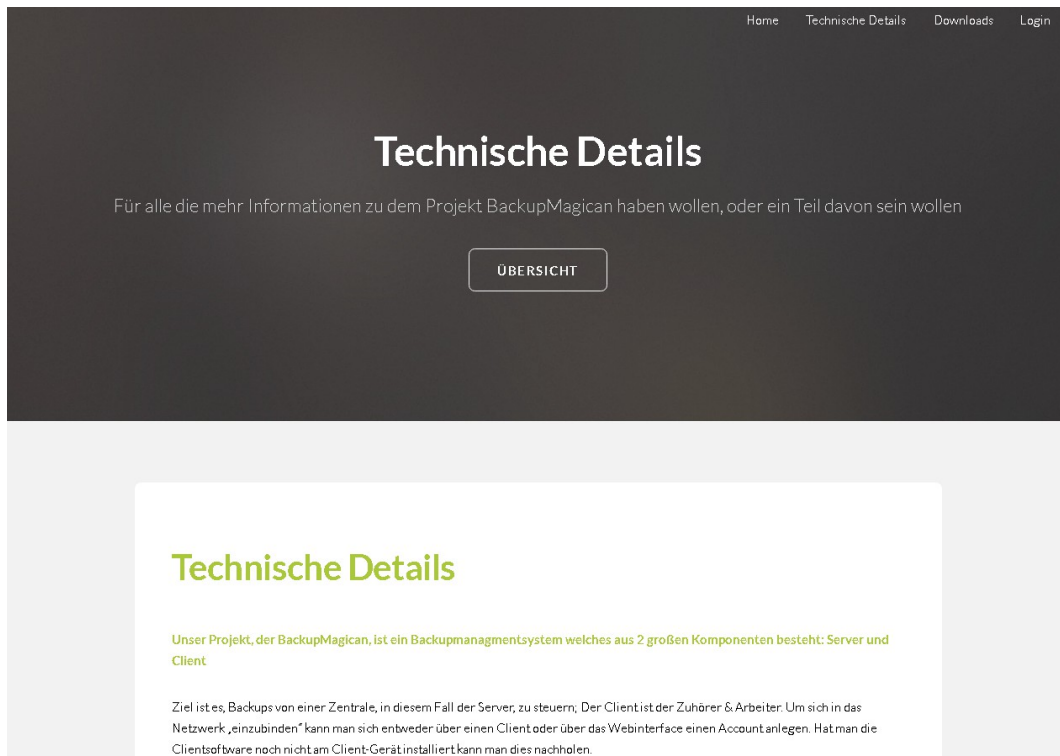


Abbildung 12: Screenshot der Website (3)

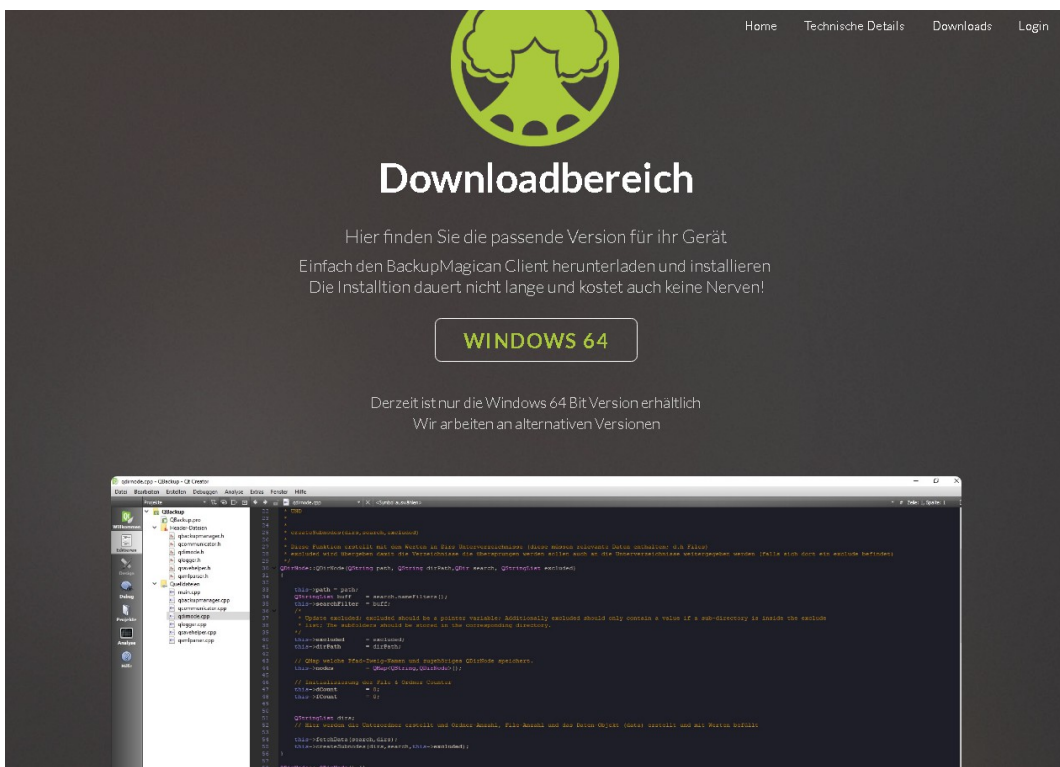


Abbildung 13: Screenshot der Website (4)



Anmeldung

Registrierung

Email*

Password*

[Passwort vergessen?](#)

ANMELDEN




Abbildung 14: Screenshot der Website (5)



Erstellung des Anmeldungs- und Registrierungs-Formulars

Als erster Schritt wurden eine ganze Reihe von Layouts für das Anmeldungs- und Registrierungs-Formular vorskizziert, um uns darauf zu einigen, welches Design und welche Benutzer-Führung unseren Vorstellungen am besten entsprechen. Es kamen dabei mehrere interessante Ideen in Frage: Es boten sich die Möglichkeiten, das Anmeldungs-Formular unabhängig vom Registrierungs-Formular als separate Webseiten zu realisieren, sowohl das Anmeldungs- als auch das Registrierungs-Formular parallel auf einer einzigen Webseite anzuzeigen oder eine einzige Webseite dafür zu nutzen, je nach Wahl des Benutzers, entweder das Anmeldungs- oder das Registrierungs-Formular anzuzeigen. Es wurden zu Testzwecken für jede dieser Möglichkeiten einfache Webseiten realisiert und zuletzt fiel die Wahl auf die scheinbar kompakteste, simpelste und zugleich benutzerfreundlichste Variante: Die, in der dem Benutzer per JavaScript, je nach Wahl, auf ein- und derselben Seite entweder die Anmeldungs- oder die Registrierungs-funktionalität geboten wurde.

Die eben beschriebene Webseite wurde mittels Notepad++ erstellt, einem kleinen Texteditor vom Software-Entwickler Don Ho, ähnlich dem gewöhnlichen, mit Windows vorinstallierten Notepad, jedoch einen deutlich größeren Funktionalitätsumfang bietend: Drag 'n' Drop, Auto-Vervollständigung, Syntax-Hervorhebung entsprechend der Dateiendung der editierten Datei, Multi-Ansicht und Plug-Ins. Dieses Programm ist einer der kleinsten und gleichzeitig beliebtesten Editoren zum Verfassen von Code in diversen Programmiersprachen oder um Webseiten von Grund auf zu erstellen. Obwohl prinzipiell leicht in der Handhabung, bietet Notepad++ keine Webseiten-Templates zur leichten Überarbeitung und so erforderte



fast jeder Schritt in der Umsetzung des Designs oder der Funktionalität eine eigene Internet-Recherche.

Es wurde eine Div-Box (Abbildung 15), die zwei weitere Div-Boxen (Abbildung 16) enthält erstellt, eine für das Anmeldungs-Formular und eine für das Registrierungs-Formular. Beide Div-Boxen erhielten dabei eigene IDs, damit sie über die angehängte CSS-Datei bearbeitet werden können.

```
<body>
  <div class="last">
    <div class="form">
      <ul class="tab-group">
        <li class="tab active"><a href="#login">Anmeldung</a></li>
        <li class="tab"><a href="#signup">Registrierung</a></li>
      </ul>

      <div class="tab-content">
        <div id="login">
          </div>

        <div id="signup">
          </div>
        </div><!-- tab-content -->
      </div> <!-- /form -->

      <div class="img-container">
        
      </div>
    </div>

    <script src='js/jquery.min.js'></script>
    <script src="js/index.js"></script>
  </body>
```

Abbildung 15: Übergeordnete Div-Box



Abbildung 16 zeigt die Anmeldungs-Div-Box (<div id="login">), die das Anmeldungs-Formular (<form [...]>) enthält. Das Anmeldungs-Formular ist zu diesem Zeitpunkt noch nicht mit einer php-Datei verknüpft, welche zukünftig den Zugriff auf die Datenbank durchführen sollte. Es sind in Abbildung 16 auch die Definitionen der in den Formularen enthaltenen Eingabefelder und Buttons zu sehen, über die der Benutzer seine Eingaben tätigen und bestätigen kann – je nach Wahl des Benutzers die des Anmeldungs- oder die des Registrierungs-Formulars.

```
<div class="tab-content">

  <div id="login">

    <form action="/" method="post">

      <div class="field-wrap">
        <label>Email<span class="req">*</span></label>
        <input type="email" required autocomplete="off" />
      </div>

      <div class="field-wrap">
        <label>Passwort<span class="req">*</span></label>
        <input type="password" required autocomplete="off" />
      </div>

      <p class="forgot">
        <a href="#">Passwort vergessen?</a>
      </p>

      <button class="button button-block">Anmelden</button>

    </form>
  </div>

  <div id="signup">

    <form action="/" method="post">

      <div class="top-row">
        <div class="field-wrap">
          <label>Vorname<span class="req">*</span></label>
          <input type="text" required autocomplete="off" />
        </div>

        <div class="field-wrap">
          <label>Nachname<span class="req">*</span></label>
          <input type="text" required autocomplete="off" />
        </div>
      </div>

      <div class="field-wrap">
        <label>Email<span class="req">*</span></label>
        <input type="email" required autocomplete="off" />
      </div>

      <div class="field-wrap">
        <label>Passwort<span class="req">*</span></label>
        <input type="password" required autocomplete="off" />
      </div>

      <button type="submit" class="button button-block">Registrieren</button>

    </form>
  </div>

</div><!-- tab-content -->
```

Abbildung 16: Div-Boxen für Anmeldung und Registrierung



```
*{
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}

html {
  overflow-y: scroll;
}

body {
  background: #D5EACD;
  font-family: 'Titillium Web', sans-serif;
}

a {
  text-decoration: none;
  color: #89c400;
  -webkit-transition: .5s ease;
  transition: .5s ease;
}

a:hover {
  color: #179b77;
}

.last {
  position: relative;
  background: #305400;
  max-width: 600px;
  height: 800px;
  margin: auto;
  margin-top: 100px;
}

.form {
  background: #305400;
  padding: 40px;
}

.tab-group {
  list-style: none;
  padding: 0;
  margin: 0 0 40px 0;
}

.tab-group:after {
  content: "";
  display: table;
  clear: both;
}

.tab-group li a {
  display: block;
  text-decoration: none;
  padding: 15px;
  background: rgba(160, 179, 176, 0.25);
  color: #a0b3b0;
  font-size: 20px;
  float: left;
  width: 50%;
  text-align: center;
  cursor: pointer;
  -webkit-transition: .5s ease;
  transition: .5s ease;
}
```

Abbildung 17: CSS-Datei

Das CSS-File (Abbildung 17) enthält Informationen, die die graphischen Eigenschaften der Elemente der Oberfläche bestimmen. Um die Box auf die gewünschte Art darstellen zu können wird eine Webkit-Erweiterung mit Präfix verwendet. Mit Hilfe der Definition „list-style: none;“ wird aus der Tabelle „.tab-group“ eine waagrechte Liste gemacht und mit Hilfe der Code-Blöcke in „.tab-group:after“ und „.tab-group li a“ wird diese Liste so eingestellt, dass entweder der Button „Anmeldung“ oder der Button „Registrierung“ farblich betont sind, je nach dem welches Formular der Benutzer gerade vor sich hat.

Die Code-Blöcke in „.form“ beziehen sich auf die jeweilige Div-Box und legen für diese die gewünschte Farbe und die gewünschten Abstände fest.



Zur Anwendung kommen zwei JavaScript-Dateien, ein selbsterstelltes JavaScript (Abbildung 18) und ein angefügtes jquery.min JavaScript, um so die eventuelle Nutzung von jQuery-Codes zu ermöglichen.

```
$( '.form' ).find( 'input, textarea' ).on( 'keyup blur focus', function (e) {  
    var $this = $(this),  
        label = $this.prev( 'label' );  
  
    if (e.type === 'keyup') {  
        if ($this.val() === '') {  
            label.removeClass( 'active highlight' );  
        } else {  
            label.addClass( 'active highlight' );  
        }  
    } else if (e.type === 'blur') {  
        if( $this.val() === '' ) {  
            label.removeClass( 'active highlight' );  
        } else {  
            label.removeClass( 'highlight' );  
        }  
    } else if (e.type === 'focus') {  
  
        if( $this.val() === '' ) {  
            label.removeClass( 'highlight' );  
        }  
        else if( $this.val() !== '' ) {  
            label.addClass( 'highlight' );  
        }  
    }  
});  
  
$( '.tab a' ).on( 'click', function (e) {  
    e.preventDefault();  
  
    $(this).parent().addClass( 'active' );  
    $(this).parent().siblings().removeClass( 'active' );  
  
    target = $(this).attr( 'href' );  
  
    $( '.tab-content > div' ).not( target ).hide();  
  
    $(target).fadeIn(600);  
});
```

Abbildung 18: Selbsterstellte JavaScript-Datei

Der obere Abschnitt des Codes definiert die Eingabefelder für den Benutzer. In diesen Eingabefeldern steht jeweils bereits ein Wort, das dem Benutzer als Hinweis dient, welche Art von Eingabe von ihm erwünscht wird. Sobald der Benutzer in ein Eingabefeld klickt,

bewegt sich dieses Hinweis-Wort unter das Eingabefeld und verkleinert sich etwas. Verlässt der Benutzer das Eingabefeld, ohne etwas eingegeben zu haben, so bewegt sich das Hinweis-Wort wieder zurück an seinen ursprünglichen Platz und nimmt seine ursprüngliche Größe an; außerdem wird der Rahmen des Eingabefeldes rot gefärbt, um dem Benutzer zu signalisieren, dass noch eine Eingabe von ihm erwartet wird. Der untere Abschnitt des Codes führt den Wechsel vom Anmeldungs- zum



Registrierungs-Formular und umgekehrt durch: Befiehlt der Benutzer den Wechsel auf das andere Formular wird das zuvor angezeigte Formular zerstört und das andere Formular an seiner Stelle erzeugt.

Erstellung einer Datenbank

Nach erfolgreicher Erstellung der Webseite zur Registrierung und zum Login war der nächste Schritt die Erstellung einer Datenbank, in der die Anmeldungsdaten der Benutzer gespeichert und abgefragt werden können. Um einen Überblick zu gewinnen wurde zunächst ein Entity-Relationship-Modell skizziert (Abbildung 19). Dann wurde überlegt, ob eine SQL-Datenbank genutzt werden sollte, oder eine Datenbank ohne SQL und die Wahl fiel vorerst auf eine SQL-Datenbank. Die Idee war, diese Datenbank so lange zu nutzen, bis eine bessere Lösung zur Speicherung und Abfrage der Benutzerdaten gefunden würde.

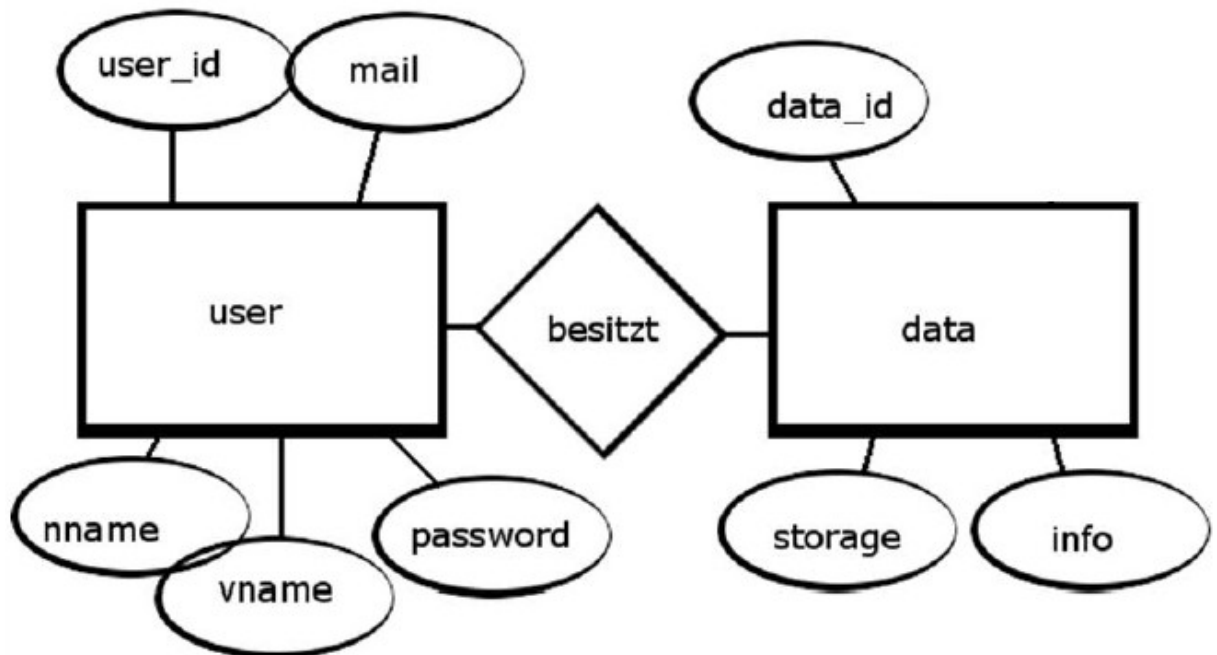


Abbildung 19: Entity-Relationship-Modell der Datenbank



Da wir im Zuge des Unterrichts bereits Erfahrungen mit „MySQL“ gesammelt hatten fiel zur Realisierung der SQL-Datenbank unsere Wahl auf diese Software. Genauer ist MySQL eine Anwendung von dem Software-Paket XAMPP, welches von dem gemeinnützigen Projekt Apache Friends entwickelt und bereitgestellt wurde. Die Software XAMPP ist wiederum eine kostenlose und einfache Apache-Distribution, welches MariaDB, PHP und Perl mit einschließt. Jedoch sollte es noch erwähnt werden, das für XAMPP in einem überaus schnellen Tempo immer wieder neue Versionen auftreten, die man zwar nicht benutzen muss, aber wenn man diese braucht, man die alte Version dann aufgeben muss. MySQL soll als Simulierung eines Datenbank-Servers auf den lokalen Rechner dienen, kann jedoch auch auf der globale Ebene erweitert werden, wobei man beachten sollte, dass die Sicherheit der übertragenen Daten hierbei nicht gewährleistet wird. XAMPP hat das Ziel, die Datenbankverwaltung und das Datenbankverständnis an Neuankömmlinge einfach und schnell zu übermitteln. Außerdem ist XAMPP eher an Entwickler gerichtet, die möglichst schnell ein kompaktes Testsystem aufsetzen wollen. Sie ist nicht zum Gebrauch als Produktivsystem (zB. öffentlicher Webserver) gedacht und es ist daher weniger wünschenswert sie als ein solches anzusehen.

Als Benutzer-Interface bietet „XAMPP“ das „Control Panel“, aus welchem ein oder mehrere Applikationen ausgewählt werden können. Über dieses wurden zuerst „Apache“ und anschließend „MySQL“ gestartet. Mittels MySQL wurde eine Test-Datenbank „db_test“ erstellt, da es sich hierbei ja auch um eine Test-Datenbank handelt. In dieser Datenbank (Abbildung 20) wurden, dem Entity-Relationship-Modell entsprechend, zwei Tabellen hinzugefügt: Die Tabelle „user“ (Abbildung 21), in der die die Benutzer



betreffenden Konto-Informationen gespeichert werden und die Tabelle „data“ (Abbildung 22) in der die Dateien der betreffenden Benutzer gespeichert werden.

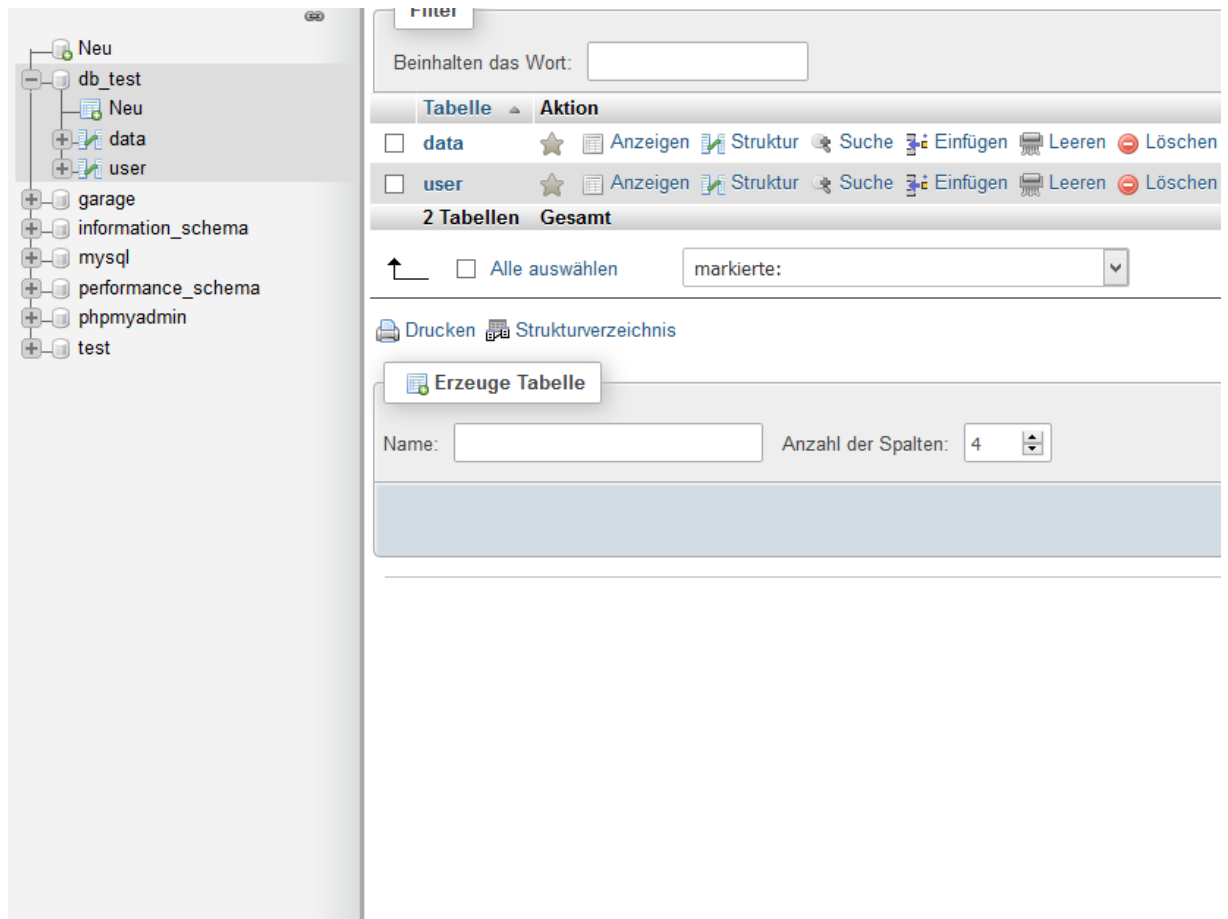
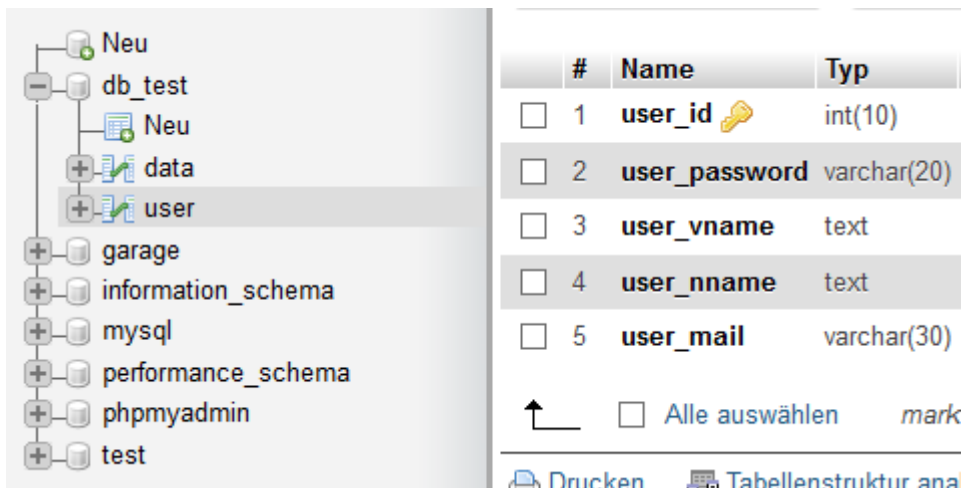


Abbildung 20: Die Tabellen der Datenbank



Die Tabelle „user“ wurde gemäß dem Entity-Relationship-Modell erstellt:



#	Name	Typ
<input type="checkbox"/> 1	user_id	int(10)
<input type="checkbox"/> 2	user_password	varchar(20)
<input type="checkbox"/> 3	user_vname	text
<input type="checkbox"/> 4	user_nname	text
<input type="checkbox"/> 5	user_mail	varchar(30)
<input type="checkbox"/>	Alle auswählen	<i>mark</i>

[Drucken](#) [Tabellenstruktur anzeigen](#)

Abbildung 21: Die Tabelle "user"

Das Feld „user_id“ in der „user“ Tabelle dient der eindeutigen Identifikation jedes Benutzers. Die Option „auto-increment“ bewirkt hier, dass jeder neu registrierte Benutzer eine „user_id“ bekommt, die um 1 höher als der zuletzt vergebene „user_id“-Wert ist. Um einen problemlosen Ablauf des Loginvorganges zu gewährleisten sollte für jeden registrierten Benutzer auch tatsächlich nur ein Datensatz in der Datenbank existieren – redundante Datensätze sollen hier unbedingt vermieden werden. In den Feldern „user_mail“ und „user_password“ werden die Strings gespeichert, die der Benutzer bei der Anmeldung als E-Mail-Adresse und Passwort angegeben hat und mit Hilfe derer er sich am Webinterface als registrierter Benutzer anmelden kann. Die Felder „user_vname“ und „user_nname“ dienen der Speicherung von Informationen für ein Benutzer-Profil, das eventuell noch realisiert wird.



#	Name	Typ	Kollation
<input type="checkbox"/> 1	data_id	int(10)	
<input type="checkbox"/> 2	user_mail	varchar(30)	latin1_swedish_ci
<input type="checkbox"/> 3	data_info	text	latin1_swedish_ci
<input type="checkbox"/> 4	data_store	longblob	

☐ Alle auswählen markierte: Anz

Drucken Tabellenstruktur analysieren

1 Spalte(n) einfügen nach data sto

Abbildung 22: Die Tabelle "data"

Auch die Tabelle „data“ wurde gemäß dem Entity-Relationship-Modell erstellt: In diesem Fall dient das Feld „data_id“ als Primärschlüssel während das Feld „user_mail“ der Verknüpfung mit der Tabelle „user“ dient. Das Feld „data_info“ enthält eine Textdatei mit zusammenfassenden Informationen über die im Feld „data_store“ gespeicherten Dateien. „data_store“ ist als „longblob“ definiert, einem Datentyp, der es ermöglicht, in einen Datenstrom umgewandelte Dateien abzuspeichern. In umgekehrter Richtung wandelt das außenstehende Empfangsprogramm den Datenstrom wieder in seine Ursprungsdateien um. Neben dem „longblob“ existieren auch noch die Datentypen „blob“ und „tinyblob“, welche die gleiche Funktion erfüllen, jedoch weniger Speicherplatz bieten. Das Speichern der Dateien hätte noch auf andere Wege realisiert werden können, doch die eben beschriebene Methode erschien als ausreichend.



Verknüpfung des Formulars mit der Testdatenbank

Um die Verbindung zwischen Webseite und Datenbank testen zu können wurden in die Tabelle „user“ zunächst manuell einige Testwerte eingetragen. Sowohl für das Anmeldungs-, als auch für das Registrierungsformular wurde je eine php-Datei erstellt: login.php (Abbildung 24) und register.php (Abbildung 25). Diese stellen die Verbindung zur Test-Datenbank dar und ermöglichen die Datenübertragung.

Die Verbindung zur Datenbank wird in einer separaten php-Datei hergestellt, welche von der HTML-Seite verlinkt wurde. Zu beachten ist dabei, dass das php-Tag nicht geschlossen wird. Die Datenbank-Verbindung wurde in einer zusätzlichen php-Datei erzeugt, die zur HTML-Seite verlinkt wurde. Wichtig ist dabei zu beachten, dass das php-Tag nicht geschlossen wird. Es wurden PHP-Variablen erstellt, in denen die Werte zum Verbindungsaufbau mit der Datenbank eingetragen wurden. Die Verbindung zur Datenbank wird durch die Funktion „mysqli“ aufgebaut.

```
<?php
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'db_test';
$mysqli = new mysqli($host,$user,$pass,$db) or die($mysqli->error);
```

Abbildung 23: PHP-Datei zum Verbindungsaufbau zur Datenbank



Im login.php (Abbildung 24) wurden zunächst in die erstellten Variablen die Werte der Input-Boxen aus der HTML-Seite eingefügt. Beim Einfügen des user_mail in die \$user_mail Variable und beim Vergleichen dessen Werte mit der in der Datenbank, wurde auch escape_string verwendet. Dies muss nicht dazugeschrieben werden, bietet aber zusätzlich Sicherheit und Schutz gegen mögliche SQL-Injections. SQL-Injections sind Angriffe auf eine Datenbank durch Einfügen von SQL-Codestücken in die Input-Boxen. Dadurch könnte beispielsweise eine Datenbank komplett gelöscht werden oder der Zugang zu einer Seite könnte ohne Passwort erfolgen. Im \$result wird nach den Informationen für die gewünschten user_mail gesucht. Sollte diese nicht gefunden werden, wird eine Nachricht zurückgeliefert, dass diese Mail noch nicht in der Datenbank existiert. Falls diese jedoch existiert, wird mit fetch_assoc() der ganze Array des gesuchten Wortes hergeholt. Anschließend wird das Passwort überprüft, sollte diese Barriere auch überwunden sein, werden die restlichen Werte, Vorname, Nachname und Mail vom Datenarray übernommen.

```
<?php

$user_mail = $mysqli->escape_string($_POST['user_mail']);
$result = $mysqli->query("SELECT * FROM users WHERE user_mail='$user_mail'");

if ( $result->num_rows == 0 )
{
    echo "User with that user_mail doesn't exist!";
}
else
{
    $user = $result->fetch_assoc();

    if ( $_POST['user_password'], $user['user_password'] )
    {
        $_SESSION['user_mail'] = $user['user_mail'];
        $_SESSION['user_vname'] = $user['user_vname'];
        $_SESSION['user_nname'] = $user['user_nname'];
    }
}
```

Abbildung 24: Die Datei "login.php"



Abbildung 25 zeigt die Funktionsweise für die Registrierung und somit auch der Eintragung der Daten in die Test-Datenbank. Hier werden in den erstellten php-Variablen die Werte aus den Eingabefeldern der HTML übernommen. Es wurde wieder `escape_string` verwendet, um SQL-Injections zu vermeiden. Dann wird mit `$result` analysiert, ob die `user_mail` bereits vorhanden ist. Das wissen wir, falls die Nummer der Reihe größer als 0 zurückliefert, was wiederum bedeutet, dass sie zweimal enthalten ist. Sollte dies zutreffen, wird eine Fehler-Nachricht entsendet. Andernfalls werden von der `$sql`-Variable aus in die Reihe `users` die jeweiligen Werte eingetragen. Sollte dabei ein Fehler bezüglich des falschen Formats oder ein anderer SQL-Fehler auftreten, wird wieder eine Fehler-Nachricht entsendet.

```
<?php
$user_vname = $mysqli->escape_string($_POST['user_vname']);
$user_nname = $mysqli->escape_string($_POST['user_nname']);
$user_mail = $mysqli->escape_string($_POST['user_mail']);
$user_password = $mysqli->escape_string($_POST['user_password']);

$result = $mysqli->query("SELECT * FROM users WHERE user_mail='$user_mail'") or die($mysqli->error());

if ( $result->num_rows > 0 )
{
    echo 'User with this user_mail already exists!';
}

else
{
    $sql = "INSERT INTO users (user_vname, user_nname, user_mail, user_password) "
        . "VALUES ('$user_vname', '$user_nname', '$user_mail', '$user_password')";

    else
    {
        echo 'Registration failed!';
    }
}
```

Abbildung 25: Die Datei "register.php"



Zum Schluss wurde das dann mit dem Registrieren (Abbildung 26) und Anmelden (Abbildung 27) überprüft (Abbildung 28).

The screenshot shows a registration form titled "Sign Up for Free". At the top, there are two buttons: "Sign Up" (highlighted in green) and "Log In". The form contains the following fields and labels:

- First Name: "Alex"
- Last Name: "Rus"
- Email Address: "test@mail.com"
- Set A Password: A password field with 8 dots.

At the bottom of the form is a large green button labeled "REGISTER".

Abbildung 26: Test-Registrierung

The screenshot shows a login form titled "Welcome Back!". At the top, there are two buttons: "Sign Up" and "Log In" (highlighted in green). The form contains the following fields and labels:

- Email Address: "test@mail.com"
- Password: A password field with 8 dots.

At the bottom of the form is a large green button labeled "LOG IN". A link labeled "Forgot Password?" is located to the right of the password field.

Abbildung 27: Test-Anmeldung

The screenshot shows a "Welcome" page. It displays the user's name "Alex Rus" and email address "test@mail.com". At the bottom of the page is a large green button labeled "LOG OUT".

Abbildung 28: Erfolgreicher Login



Design des Logos

Für die Erstellung des Logos war sehr viel Überlegung und Kreativität gefragt. Zum einen sollte es zeigen, dass es sich hierbei um ein Backup handeln soll, zum anderen sollte der Anblick des Symbols auch die Idee einer Cloud vermitteln. Somit kam eine Wolke mit mehreren Verbindungen zustande (Abbildung 29).



Abbildung 29: Erste Variante des Logos

Es fehlte jedoch die Farbeneinheitlichkeit und das Design stellte sich eher als altmodisch als modern dar, weshalb die Inspiration in Programmen mit ähnlichen Zwecken gesucht wurde. Als bestes Beispiel einer Backup-Cloud wurde dann die Dropbox genommen, da diese sehr benutzerfreundlich gestaltet wurde und deren Emblem auch nicht unbemerkt blieb. Fokussiert



wurde eher auf deren Design, dass es simpel jedoch bemerkenswert aus der Masse herausstach. Von daher kam die Inspiration, das Design leicht zu verändern und es so moderner darstellen zu lassen. Grundsätzlich wurde am Hauptdesign nichts verändert, jedoch wurde die Farbe einheitlich gestaltet und die Darstellung anstelle verschiedener Farben mit weißen Konturen verziert.

Es hat noch sehr viele Versuche und Kombinationen gebraucht, vor allem im Farbbereich, doch nach unzähligen Abstimmungen kamen wir schlussendlich zum gewünschten Ergebnis (Abbildung 30). Lustigerweise kam beim Bild nicht nur eine Wolke mit Verbindungen als Darstellung heraus, das ja bekanntlich eine Cloud symbolisiert, sondern sie stellt auch einen Baum mit verankerten Wurzeln dar, welches für ein sicheres Backup stehen könnte.



Abbildung 30: Finales Logo



Design der Icons

Verglichen mit dem Logo, welches das Projekt ja symbolisiert, waren die Icons eher was experimentelles und flüchtiges. Noch ungewiss, ob wir überhaupt Icons für die Dateien im Programm verwenden oder sie einfach im standardmäßigen Format lassen, kamen wir zum Entschluss, dass es zumindest eigene Icons gäben müsste, wenn nicht für den Moment, aber dann in naher Zukunft. Obwohl diese Arbeit leichter fiel als die Erstellung des Logos, war sie nicht gerade von kurzer Dauer. Trotz des immer verwendeten Grundrisses, dem File, mussten die Symbole für die jeweiligen Formate genauestens überlegt werden. Sie mussten original von null aus designt werden und durften nicht einfach schlampig dargestellt werden. Für den Anfang sind wir der Meinung, dass zwischen Musik, Bilder, Videos, Text, Programm, Archiv und Ordner unterscheidet werden soll, alles andere wird als Leer-File dargestellt (Abbildung 31).

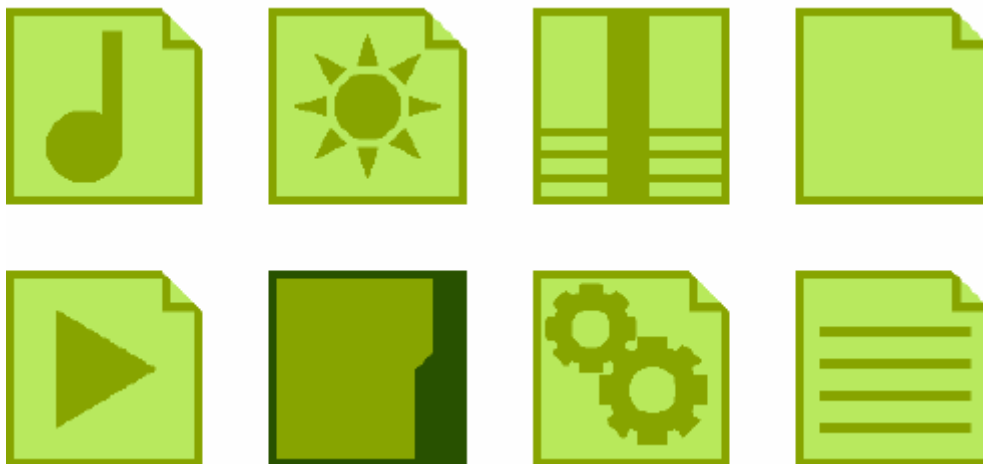


Abbildung 31: Erste Variante der Icons



Eventuell wurden auch die Icons überarbeitet, da diese zum Design des Emblems dazugehören müssen (Abbildung 32).

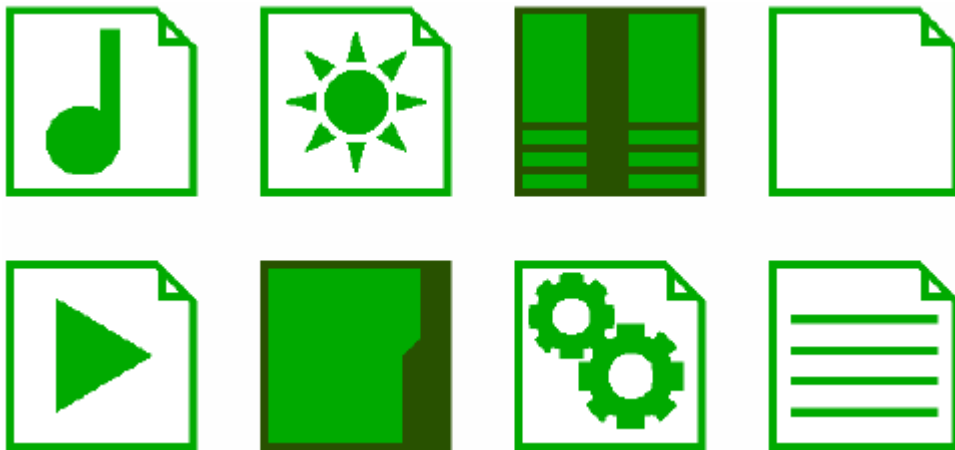


Abbildung 32: Finale Icons

Arbeitsaufteilung:

Erstellung des BackupMagician Server Programms	Polak, Morhammer, Rus
Erstellung der Testdatenbank	Rus
Erstellung der Website	Polak, Rus
Verbindung der Website mit Testdatenbank und Server	Rus
Design des Logos und der Icons	Rus
Erstellung des Webtemplate	Rus, Polak
Erstellung des Kommunikators	Morhammer
Erstellung des Clients (Python Skript, BackupMagician Setup und BackupMagician Core)	Polak



Abbildungsverzeichnis

Abbildung 1: Die Klassen des BackupMagician Core Programms.....	12
Abbildung 2: Die readyRead-Funktion der MyThread Klasse des Servers.....	15
Abbildung 3: Flowchart des BackupMagician Setup.....	20
Abbildung 4: Flowchart des BackupMagician Core Programms.....	22
Abbildung 5: Aufbau des BackupMagician.....	23
Abbildung 6: BackupMagician Setup - Anmeldung und Registrierung.....	26
Abbildung 7: BackupMagician Setup - Ordnerauswahl.....	27
Abbildung 8: BackupMagician Setup - Cloudspeicher Zugangsdaten.....	29
Abbildung 9: BackupMagician Setup - Hauptbildschirm.....	30
Abbildung 10: Screenshot der Website (1).....	36
Abbildung 11: Screenshot der Website (2).....	36
Abbildung 12: Screenshot der Website (3).....	37
Abbildung 13: Screenshot der Website (4).....	37
Abbildung 14: Screenshot der Website (5).....	38
Abbildung 15: Übergeordnete Div-Box.....	40
Abbildung 16: Div-Boxen für Anmeldung und Registrierung.....	41
Abbildung 17: CSS-Datei.....	42
Abbildung 18: Selbsterstellte JavaScript-Datei.....	43
Abbildung 19: Entity-Relationship-Modell der Datenbank.....	44
Abbildung 20: Die Tabellen der Datenbank.....	46
Abbildung 21: Die Tabelle "user".....	47
Abbildung 22: Die Tabelle "data".....	48
Abbildung 23: PHP-Datei zum Verbindungsaufbau zur Datenbank.....	49
Abbildung 24: Die Datei "login.php".....	50
Abbildung 25: Die Datei "register.php".....	51
Abbildung 26: Test-Registrierung.....	52
Abbildung 27: Test-Anmeldung.....	52
Abbildung 28: Erfolgreicher Login.....	52
Abbildung 29: Erste Variante des Logos.....	53
Abbildung 30: Finales Logo.....	54
Abbildung 31: Erste Variante der Icons.....	55
Abbildung 32: Finale Icons.....	56