

Backup Magican

Inhalt

Einführung.....	2
Projekt Zusammenfassung.....	3
Projektdefinition.....	4
Ziele, Nicht-Ziele und Milestones.....	5
System Overview.....	6
System Komponenten Beschreibung.....	6
Hardware Komponenten.....	6
Software Komponenten.....	6
Der Kommunikator.....	6
System Design.....	8
Technologie Auswahl.....	8
Systemarchitektur.....	9
System Implementierung.....	9
Entwicklung des Kommunikators.....	9
Client Software.....	10
BackupMagican Setup.....	10
Aufbau des BackupMagican Setup.....	10
BackupMagican Core.....	17
1. Task verarbeiten.....	17
2. Ordner- / Filestruktur anlegen.....	18
3. Task verarbeiten.....	18
Server Software.....	19
Web Monitoring Software.....	19
Erstellung des Anmeldungs- und Registrierungs-Formulars.....	19
Erstellung einer Datenbank.....	25
Verknüpfung von Formular mit der Testdatenbank.....	29
Logo und Icons designen.....	30
Erstellen einer Webinterface.....	30

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 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 „Backupmagan“ 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.

Projekt Zusammenfassung

Diplomarbeit: **Cloudbasiertes Backup-System**

Projektnummer: ___ BI 17/18

Jahrgang: 5/6 BBKIF

Schuljahr: 2017/18

Kandidaten: Roland Morhammer

Rene Polak

Richard Rus

Betreuer: Prof. Dipl. Ing. Endre Beda

Prof. Ing. Mag. Manfred Oberkersch

Projektdefinition

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 man sich 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.

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 (Professionelle Ansätze):

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										

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 Client dazu nur vorübergehend eine Verbindung mit Hilfe des QTcpSocket-Objekts auf, die lediglich der einmaligen Übermittlung eines einzelnen JSon-Dokuments dient.

Der Kommunikator ist zur Abwicklung der Kommunikationserfordernisse folgender Aufgaben notwendig:

1. Registrierung eines neuen Clients beim Server
2. Anmeldung eines registrierten Clients beim Server
3. Registrierung eines neuen Benutzers beim Server
4. Anmeldung eines registrierten Benutzers beim Server
5. Übermittlung eines Backup-Tasks vom Server an den Client

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

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. Wie die Implementierung dieser Routinen realisiert wurde ist unter der Überschrift „Der Kommunikator“ im Teil „Software Komponenten“ dieser Diplomprojektdokumentation beschrieben.

Client Software

BackupMagician Setup

Eines der Programme, die es dem Anwender ermöglicht 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

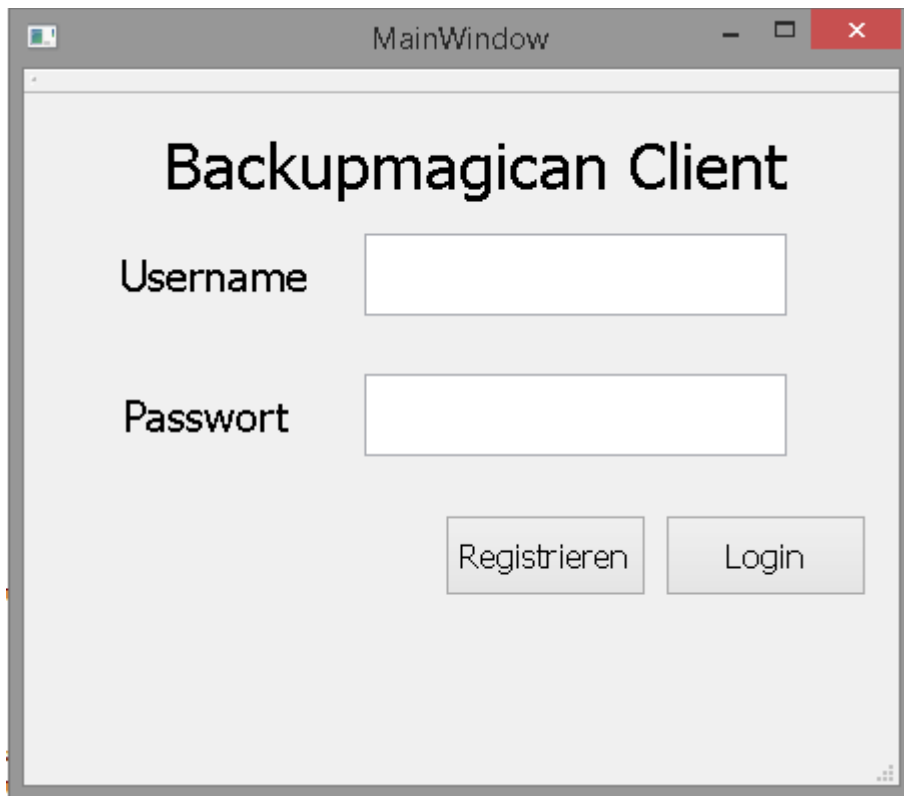
Aufbau des BackupMagician Setup

Der Client ist in 4 wichtige Abschnitte unterteilt:

1. Anmeldung 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



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



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 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.

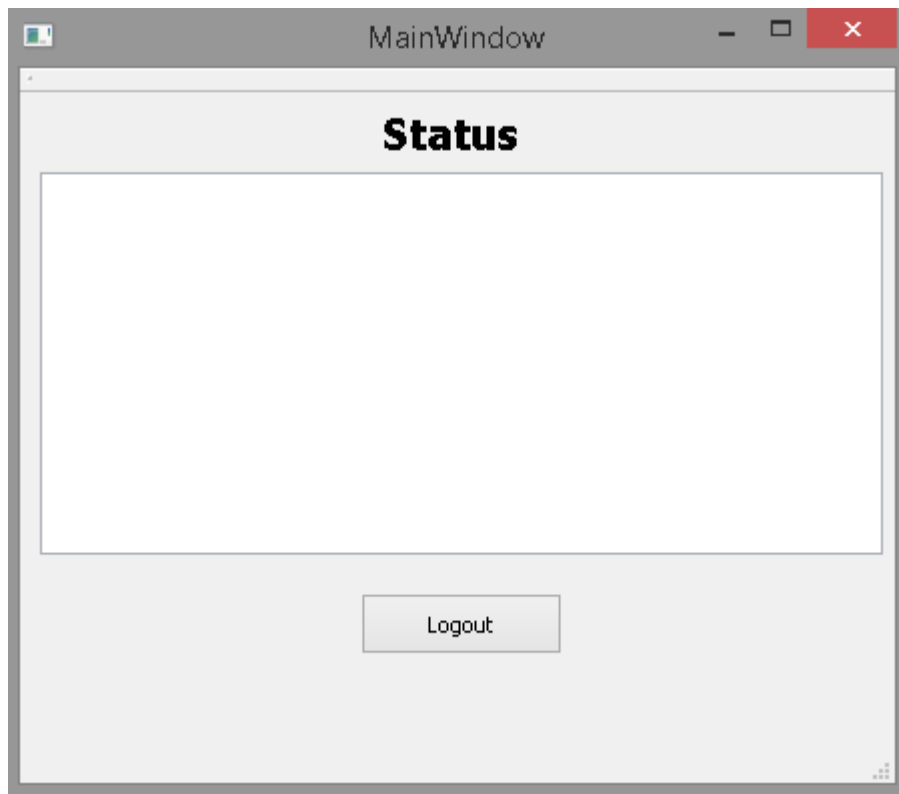
3. Setup Cloudspeicher Zugangsdaten

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.

4. 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

Der BackupMagician Core bekommt Aufgaben in JSON-Notation vom BackupMagician Setup, welcher die Aufgaben, die der Server an ihn schickt an den BackupMagician Core 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-Script. Dieses Python-Script kommuniziert dann direkt mit Dropbox und lädt die zu sichernden Dateien Datei für Datei hoch.

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

Server Software

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.

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 Registrierungsfunctionalität 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 [Abb.1], die zwei weitere Div-Boxen [Abb.2] enthält, erstellt, eine für das Anmeldungs-Formular und eine für das

```
<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>
```

Registrierungs-
Formular. Beide Div-
Boxen erhielten
dabei eigene IDs,
damit sie über die
angehängte CSS-
Datei bearbeitet
werden können.

[Abb.1]

Abbildung 2 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 2 auch die Definitionen der in den Formularen enthaltenen Eingabefelder und Buttons zu sehen, über 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 -->
```

Das CSS-File 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.

[Abb.2]

[Abb.3]

Zur Anwendung kommen zwei JavaScript-Dateien, ein selbsterstelltes JavaScript [Abb. 4] 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);
});

```

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

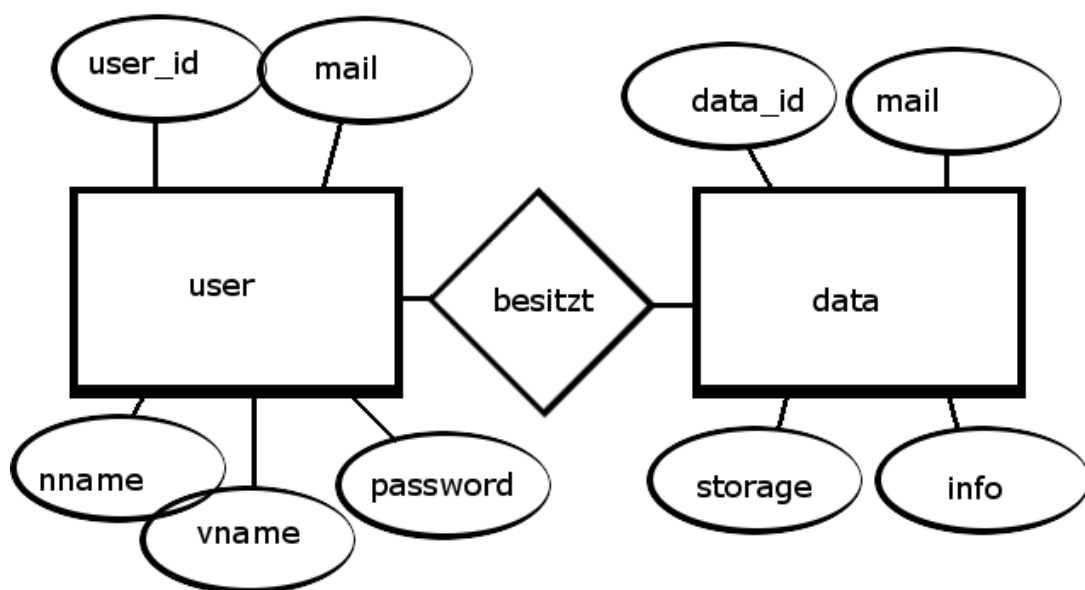
Benutzer in ein Eingabefeld klickt,

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.

[Abb.4]

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. [Abb. 5] 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.

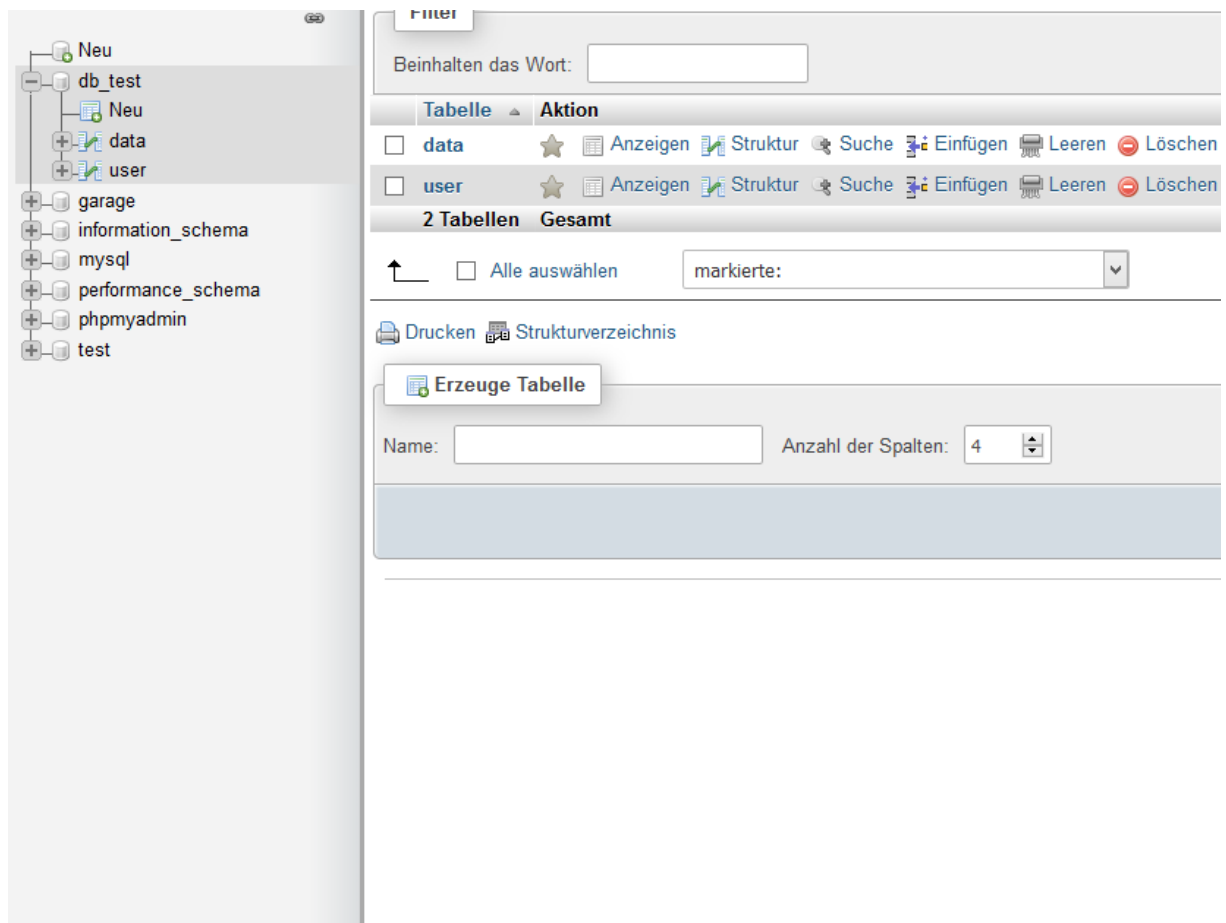


[Abb.5]

Da wir im Zuge des Unterrichts bereits Erfahrungen „XAMPP“ gesammelt hatten fiel zur Realisierung der MySQL-Datenbank unsere Wahl auf dieses Software-Paket, welches den Webserver „Apache“, die Datenbanklösung „MariaDB“, und die Skriptsprachen „PHP“ und „Perl“ enthält. Mit Hilfe von „XAMPP“ wurde zunächst ein Datenbank-Server lokal in Betrieb

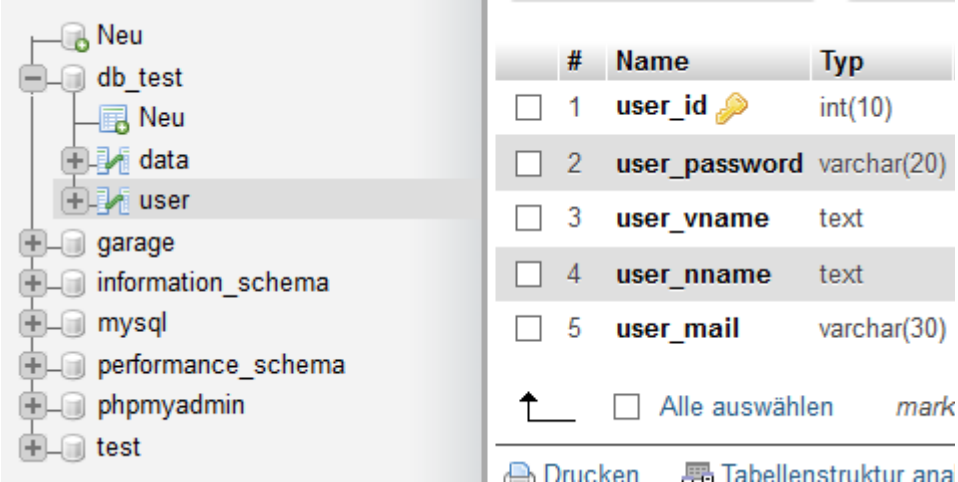
genommen, wozu sich dieses Software-Paket besonders gut eignet. Zur Nutzung von „XAMPP“ in einem Produktivsystem empfiehlt sich eine umfassende Änderung der „XAMPP“-Konfiguration, um einen einigermaßen sicheren Betrieb zu ermöglichen.


Als Benutzer-Interface bietet „XAMPP“ das „Control Panel“. Über dieses wurden zuerst „Apache“ und anschließend „MySQL“ gestartet. Mittels MySQL wurde eine Test-Datenbank „db_test“ erstellt, in der, dem Entity-Relationship-Modell [Abb. 6] entsprechend, zwei Tabellen erzeugt wurden: die Tabelle „user“ [Abb. 7], in der die die Benutzer betreffenden Informationen gespeichert werden und die Tabelle „data“ [Abb. 8] in der Dateien gespeichert werden.



[Abb.6]

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)

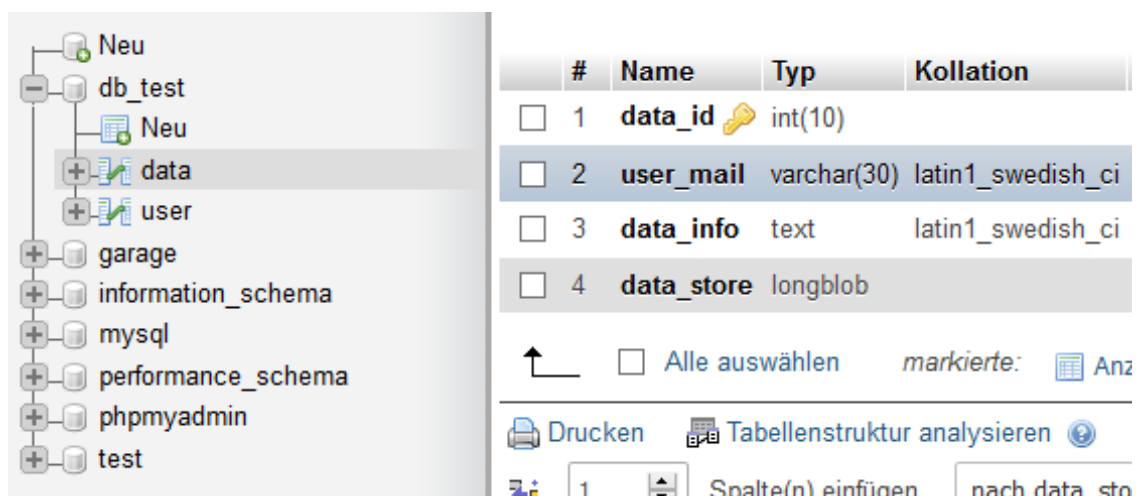
☐ [Alle auswählen](#) *mark*

[Drucken](#) [Tabellenstruktur ana](#)

[Abb.7]

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.



[Abb.8]

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 Anmeldungs-, als auch für das Registrierungsformular wurde je eine php-Datei erstellt: login.php (Abbildung 9) und register.php (Abbildung 10). Diese stellen die Verbindung zur Test-Datenbank her 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 Datenbank eingetragen wurden. Die Verbindung zum Host und zur Datenbank wurden dann mit `new mysqli` geöffnet.

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);
```

[Abb.9]

Bei Abarbeitung der Datei login.php (Abbildung 10) werden zunächst die Strings aus den Eingabefeldern der HTML-Seite in die entsprechenden Variablen geschrieben. Der String aus dem 'user_mail' Feld wird zunächst von der Funktion „\$mysqli->escape_string“ überprüft und bearbeitet, bevor er in die Variable \$user_mail geschrieben wird. Dies ist nicht unbedingt erforderlich, bietet aber eine gewisse Sicherheit vor eventuellen SQL-Injections. Eine SQL-Injection ist ein Angriff auf eine Datenbank, bei dem an eine Eingabe in ein Eingabefeld SQL Code auf eine Weise hinzugefügt wird, dass das Datenbank-System diesen Code dann ausführt. Mit solch einem Eingriff könnte beispielsweise bewirkt werden, dass die Datenbank komplett gelöscht wird oder der Hacker Zugang zu einer Seite erhält, für die er keine Berechtigung besitzt. Danach werden die Daten des Benutzers mittels der Funktion „\$mysqli->query“ abgefragt und in der Variable \$result gespeichert. Sollte der Benutzer nicht anhand der angegebenen E-Mail-Adresse gefunden werden können bleibt die Variable \$result leer, und es wird zurückgemeldet, dass der Benutzer in der Datenbank noch nicht existiert. Konnte der Benutzer gefunden werden, so liefert die Funktion „\$result->fetch_assoc“ das gesamte Ergebnis der in \$result gespeicherten Datenbankabfrage und schreibt dieses in die Variable \$user. Danach wird das Passwort überprüft und nach erfolgreicher Überprüfung werden die E-Mail-Adresse des Benutzers, sein Vorname und sein Nachname in das Session-Array (\$_SESSION) übernommen.

```
<?php

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

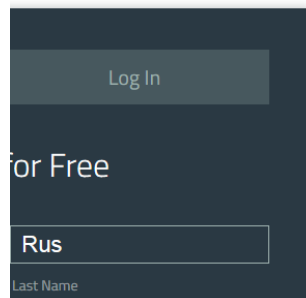
[Abb.10]

Abbildung 11 zeigt wie die Eintragung der Werte in die Test-Datenbank durch die php-Datei zur Registrierung funktioniert. Zunächst werden die Werte aus den Eingabefeldern der HTML-Seite in die entsprechenden php-Variablen geschrieben. Auch hier kam zur Vermeidung von SQL-Injections die Funktion `$mysqli->escape_string` zur Anwendung. Als nächstes wird die Datenbank mittels der Funktion „`$mysqli->query`“ nach einem Benutzer mit der angegebenen E-Mail-Adresse durchsucht und das Ergebnis in die Variable `$result` geschrieben. Wenn die Variable `$result` einen Datensatz enthält (`$result->num_rows > 0`) ist bereits ein Benutzer mit der angegebenen E-Mail-Adresse vorhanden und eine entsprechende Fehlermeldung wird ausgegeben. Andernfalls werden mittels der Variable `$sql` die entsprechenden Strings in die Tabelle „users“ der Datenbank eingetragen.

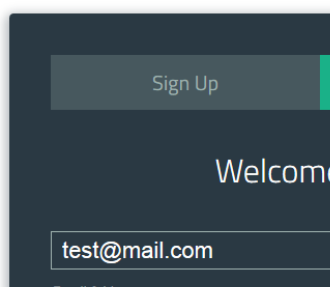
```
$mysqli->error();
```

[Abb.11]

Zum Schluss wurde das dann mit dem Registrieren [Abb.12] und Anmelden [Abb.13] überprüft [Abb.14].



[Abb.12]



[Abb.13]



[Abb.14]

Logo und Icons designen

Zur Gestaltung eines Logos für das Projekt waren sehr viel Überlegung und Kreativität gefragt. Zum einen sollte das Logo zeigen, dass es ein Backup-System repräsentiert, zum anderen sollte der Anblick des Symbols auch die Idee einer Wolke („Cloud“) vermitteln. Dies führte zum Design einer Wolke, in die mehrere Verbindungen mündeten (Abbildung 15).



[Abb.15]

Zu diesem Zeitpunkt im Designprozess des Logos schien das Logo noch eher altmodisch als modern und es die Farben schienen noch zu uneinheitlich, deshalb wurde bei den Logos von Programmen mit ähnlichen Zecken wie dem des BackupMagician nach Inspiration gesucht. Ein scheinendes Beispiel für ein Cloudservice stellt, aufgrund seiner benutzerfreundlichen Oberfläche, die Dropbox dar, deren Logo auch nicht unbemerkt geblieben war. Das Logo für den BackupMagician sollte simple sein und aus der Masse herausstechen und so entstand die Idee, den ersten Entwurf des Logos leicht zu verändern und moderner zu machen. Nach etlichen Experimenten mit unterschiedlichen Konturen und Farben konnte ein zufriedenstellendes Ergebnis erzielt werden. So wurden die

Formen des Grunddesigns prinzipiell beibehalten, die Farbgebung wurde vereinheitlicht und der Form der Wolke wurde eine weiße Kontur gegeben. Interessanterweise stellte sich heraus, dass die Darstellung einer Wolke mit Verbindungen nach unten im Ergebnis einem Baum glich, der mit seinen Wurzeln fest im Boden verankert war – ein Symbol für ein sicheres Backup.



[Abb.16]

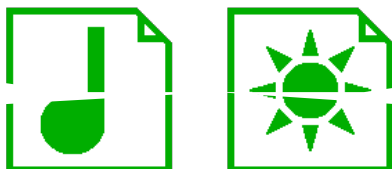
Das Hauptaugenmerk war auf das Design des Logos, dem Aushängeschild des Projekts, gefallen, jedoch sollten auch noch Icons kreiert werden, welche im Programm die verschiedenen Dateien und Ordner symbolisieren würden. Zu Beginn stand noch nicht fest, ob das Design dieser Icons überhaupt erforderlich war, nichtsdestotrotz wurden sie alsbald erstellt. Das Erstellen der Icons war nicht ganz so aufwändig wie das Design des Logos, nahm aber trotzdem einiges an Zeit in Anspruch. Als Rahmen für die verschiedenen Symbole für Dateien wurde ein Quadrat mit einer geknickten Ecke gewählt, die Ordner sollten durch eine klassische Ordner-Form repräsentiert werden. Die Symbole für die verschiedenen Arten von Dateien wollten sorgfältig überlegt sein und sollten etwas von Grund auf frisch designt werden, ohne an bereits vorhandene Designs anzulehnen. So entstanden sieben Symbole, eines zur Darstellung der Ordner und

sechs zur Darstellung verschiedener Arten von Dateien: Musik, Bilder, Videos, Text, ausführbare Dateien und Archive. Dateien welche nicht zugeordnet werden könnten sollten durch einen Rahmen ohne Inhalt dargestellt werden (Abbildung 17).



[Abb.17]

Zuletzt wurden die Icons noch einmal überarbeitet, um diese farblich an das Projekt-Logo anzupassen.



[Abb.18]

Erstellen einer Webinterface

Erstellen der Server-App