

# DIPLOMARBEIT

Backup-Magician

Cloudbasiertes Backup-System





Diplomarbeit: **Cloudbasiertes Backup-System**

Projektnummer: \_\_\_ BI 17/18

Jahrgang: 6 BBKIF

Schuljahr: 2017/18

Kandidaten: Morhammer Roland

Polak Rene

Rus Richard

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.



# Inhaltsverzeichnis

Summary1

Projektdefinition1

Zusammenfassung1

Inhaltsverzeichnis1

Einführung1

Ziele, Nicht-Ziele und Milestones8

System-Komponenten Beschreibung9

Der Kommunikator**Fehler! Textmarke nicht definiert.**

System Design11

Technologie Auswahl11

Systemarchitektur13

System Implementierung**Fehler! Textmarke nicht definiert.**

Kommunikator Entwicklung21

Client Software**Fehler! Textmarke nicht definiert.**

Backup Magician Setup21

Backup Magician Setup - Aufbau21

Backup Magician Core21



Server Software**Fehler! Textmarke nicht definiert.**

Web Monitoring Software21

Website**Fehler! Textmarke nicht definiert.**

Das Login- und Registry- Formular21

Die Test-Datenbank21

Verknüpfung von Formular mit der Testdatenbank 21

Design**Fehler! Textmarke nicht definiert.**

Logo 21

Icons21



## 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 „Backup Magican“ 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 (Professionelle Ansätze):

1. Usermanagementsystem 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-Komponentenbeschreibung

### 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). Das QTcpServer-Objekt der Server Software verwendet dazu den Port 1234, das QTcpServer-Objekt der Client Software den Port 1244. 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,



spricht je aktivem Client ein Objekt der „MyThread“ Klasse, welche 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.



## System Implementierung

### Kommunikator Entwicklung

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 Server-Dummy programmiert werden. Zuerst wurde sowohl am Client- als auch am Server-Dummy eine externe Bibliothek eingebunden: der „QtWebApp HTTP Webserver“ von [www.stefanfrings.de](http://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](http://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 Diplomprojektsdokumentation 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

MainWindow

### Backupmagician Client

Username

Passwort

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

MainWindow

## Fast geschafft

Überprüfen sie bitte ihre Eingaben!

Zusätzlich haben sie jetzt die Möglichkeit ihre Cloud Speicher Zugangsda:

Username:

Passwort:

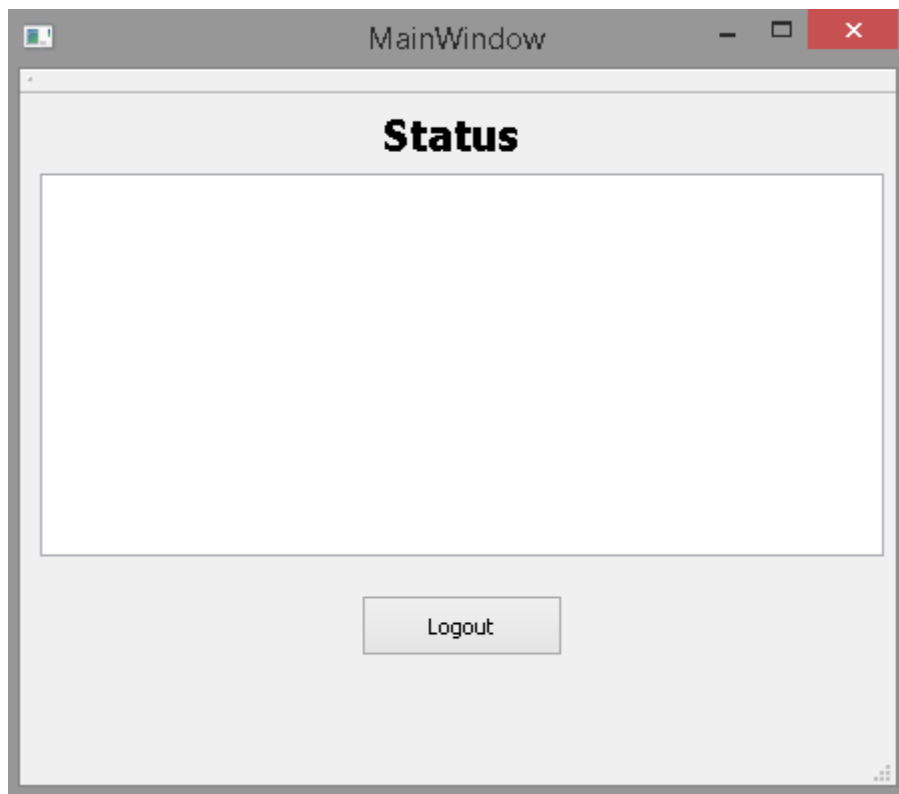
Fertig Zurück Cloud Daten hinzufügen

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 Backup-Magician 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 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.



## Website

### Das Login- und Registry- Formular

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 Anmeldeformular und das Registrierungsformular in zwei separate Webseiten zu halten, dann ergab sich die Möglichkeit, alles in einer Seite verfügbar zu machen und schlussendlich biete sich auch die Alternative an, die Anmeldung und die Registrierung in einem Fenster zu erzeugen, wobei diese sich immer nur abwechselnd zeigen würden, je nachdem, welches Formular man gerade bräuchte. Jede Option und Methode zur Erstellung der Webseite wurde zu Testzwecken realisiert und ausprobiert. Zuletzt fiel die Wahl auf die scheinbar kompakteste, simpelste und zugleich benutzerfreundlichste Variante: Die, in der dem Benutzer mittels JavaScript, je nach Wahl, auf ein- und derselben Seite entweder die Anmeldungs- oder die Registrierungsfunktionalität geboten wurde.

Die Software, welche für das Erstellen der Anmeldung und Registrierung zum Einsatz kam, nennt sich Notepad++, ein kleiner Texteditor geschrieben vom Software-Entwickler Don Ho, ähnlich dem gewöhnlichen, mit Windows vorinstallierten Notepad, jedoch einen deutlich größerer Funktionalitätsumfang bietend. Mit den Möglichkeiten wie Drag 'n' Drop, Auto-Vervollständigung, Syntax-Hervorhebung (unter Verwendung der jeweiligen Programmiersprache mit der dazugehörigen Dateiendung),



Multi-Ansicht und Plug-Ins wurde dieses Programm leicht zur einer der beliebtesten und einfachsten Tools, mit denen man zwar mehrere Programmiersprachen-orientierte Applikationen, aber vor allem Web-basierte Seiten oder Projekte von null aus erstellen kann. 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 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>
```

[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 -->
```

[Abb.2]





```
*{
  -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;
}
```

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.3]



Zur Anwendung kommen dabei zwei JavaScript-Dateien, ein selbsterstelltes JavaScript [Abb. 4] und ein angefügtes jquery.min

```
$('.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);  
});
```

[Abb.4]

JavaScript, um so das Verwenden einiger jQuery-Codes zu ermöglichen, falls es benötigt werden sollte.

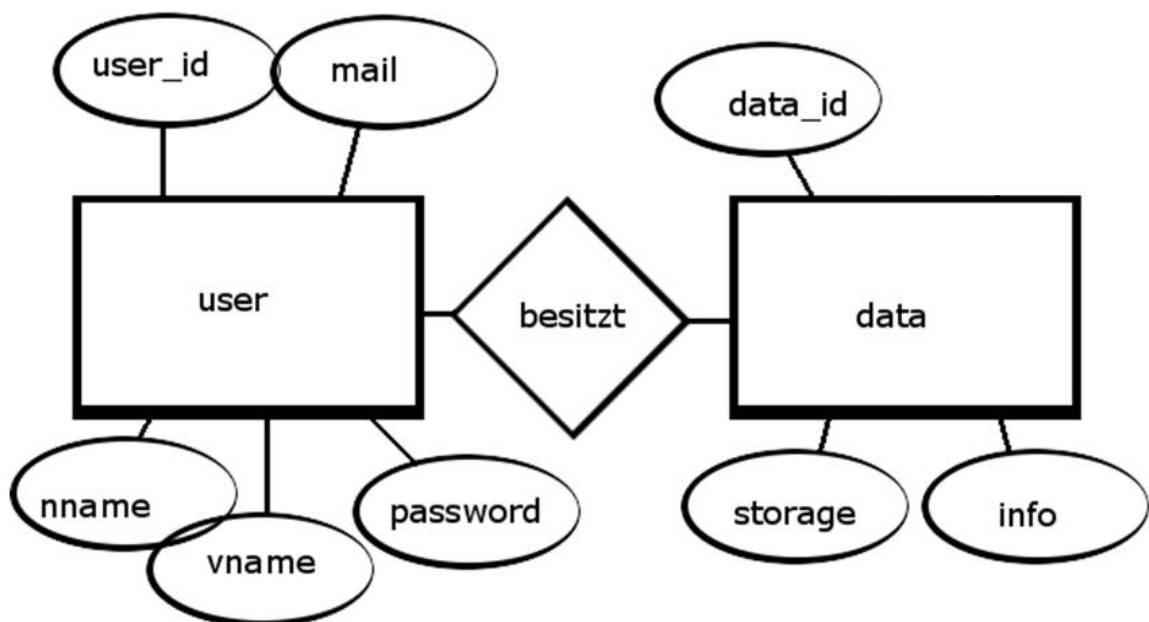
Der erste Abschnitt des Codes definiert die Eingabefelder für den Benutzer. In jedes dieser Eingabefelder ist jeweils bereits ein Wort, das dem User als Hinweis dient und angibt,

welche Art von Eingabe von ihm erwünscht wird. Sobald der User 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 mit ursprünglicher Größe; dabei wird der Feldrahmen 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.



## Die Test-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, denn ohne diese wäre ein Anmeldungs- und Registrierungsformular sinnlos. Um einen Überblick zu gewinnen wurde zunächst ein Entity-Relationship-Modell skizziert [Abb. 5]. Dies dient zu einem orientierungsvollerem Überblick über die Datenbank, seiner Tabellen und dessen Funktionsrollen. 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]

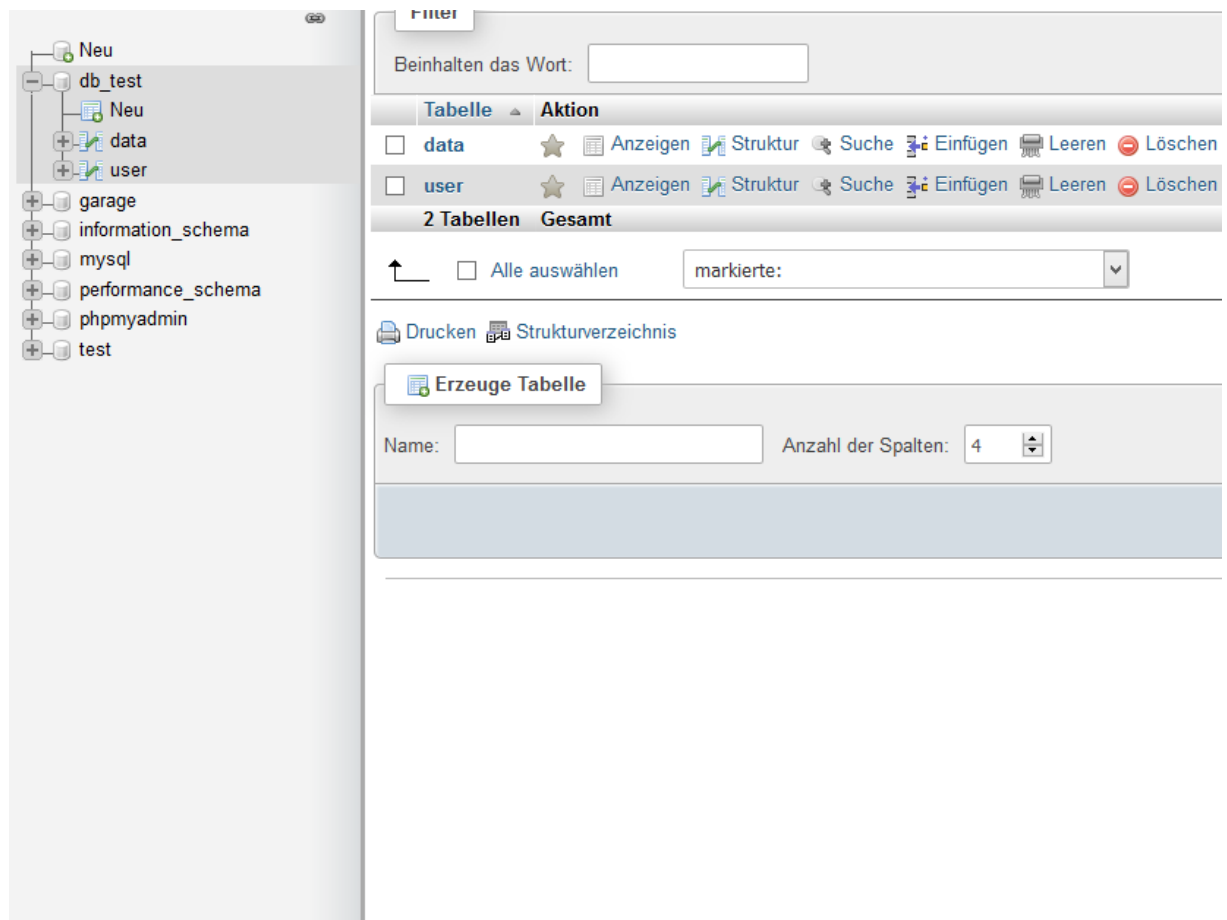


Da wir im Zuge des Unterrichts bereits Erfahrungen mit „MySQL“ gesammelt hatten fiel zur Realisierung der SQL-Datenbank unsere Wahl auf diese Software. Genauer ist MySQL eine Anwendung von dem Software-Paket XAMPP, welches von dem gemeinnützigen Projekt Apache Friends entwickelt und bereitgestellt wurde. 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, das 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 es 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 [Abb. 6] wurden, dem Entity-Relationship-Modell entsprechend, zwei Tabellen hinzugefügt: Die Tabelle „user“ [Abb. 7], in der die Benutzer betreffenden



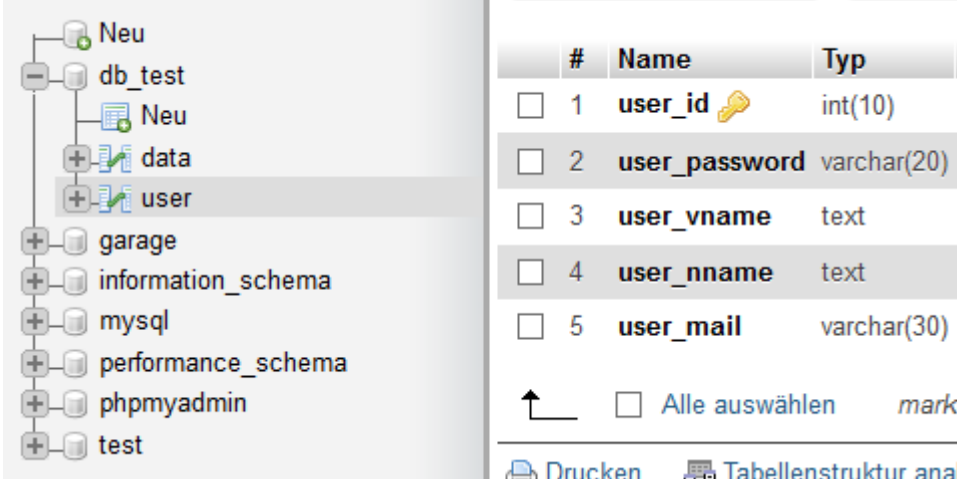
Konto- Informationen gespeichert werden und die Tabelle „data“ [Abb. 8] in der die Dateien der betreffenden Benutzer gespeichert werden.



[Abb.6]



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

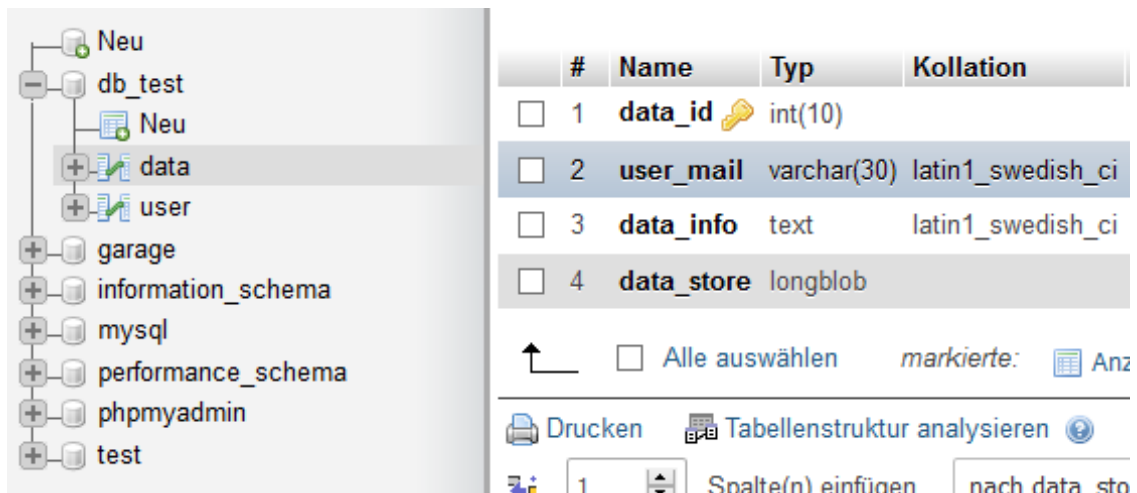


#	Name	Typ
<input type="checkbox"/> 1	<b>user_id</b>	int(10)
<input type="checkbox"/> 2	<b>user_password</b>	varchar(20)
<input type="checkbox"/> 3	<b>user_vname</b>	text
<input type="checkbox"/> 4	<b>user_nname</b>	text
<input type="checkbox"/> 5	<b>user_mail</b>	varchar(30)
<input type="checkbox"/>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

[Drucken](#) [Tabellenstruktur analysieren](#)

[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]





Im login.php [Abb.10] 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'];
    }
}
```

[Abb.10]



[Abb.11] 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 `$result` analysiert, ob die `user_mail` bereits vorhanden ist. Das wissen wir, falls die Nummer der Reihe größer als 0 zurückliefert, was wiederum bedeutet, dass sie zweimal enthalten ist. Sollte dies zutreffen, wird eine Fehler-Nachricht entsendet. Andernfalls wird von der `$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!';
    }
}
```

[Abb.11]



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

The registration form is titled "Sign Up for Free". It features a dark blue background with a teal header bar containing "Sign Up" and "Log In" buttons. The form fields include: "First Name" (containing "Alex"), "Last Name" (containing "Rus"), "Email Address" (containing "test@mail.com"), and a password field with 10 dots. A teal "REGISTER" button is at the bottom.

[Abb.12]

The login form is titled "Welcome Back!". It features a dark blue background with a teal header bar containing "Sign Up" and "Log In" buttons. The form fields include: "Email Address" (containing "test@mail.com") and a password field with 10 dots. A teal "LOG IN" button is at the bottom. A "Forgot Password?" link is visible next to the password field.

[Abb.13]

The user profile page is titled "Welcome". It features a dark blue background with a teal header bar. The user's name "Alex Rus" and email address "test@mail.com" are displayed in teal. A teal "LOG OUT" button is at the bottom.

[Abb.14]



## Design

### Logo

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[Abb.15].



[Abb.15]

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 eines Backup-Clouds wurde dann die Dropbox genommen, da diese sehr benutzerfreundlich gestaltet wurde und dessen Emblem auch nicht



unbemerkt blieb. Fokussiert wurde eher auf deren Design, das es simple 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 [Abb.16]. 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.



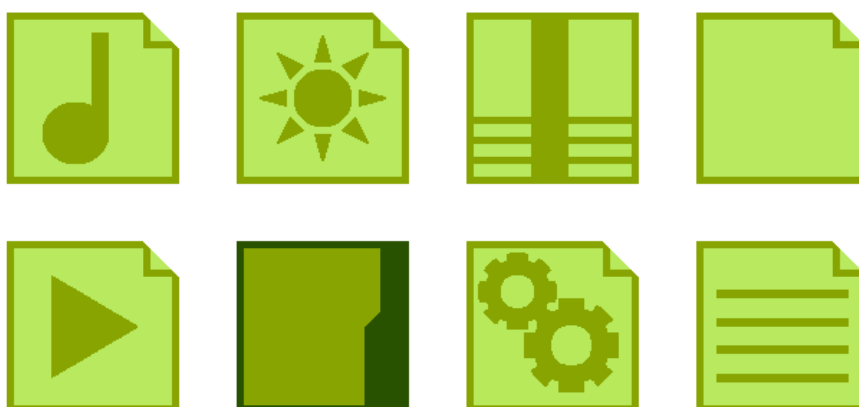
[Abb.16]



## 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, das es zumindest eigene Icons geben 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.

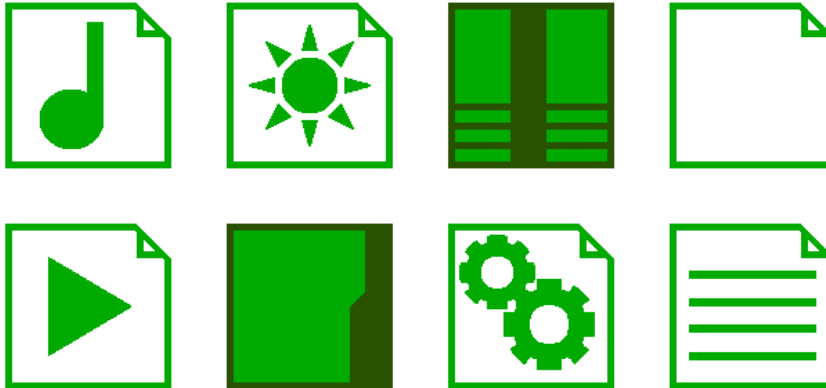
Trotz des immer verwendeten Grundrisses, dem File, mussten die Symbole für die jeweiligen Formate genauestens überlegt werden. Sie mussten original von null aus designt werden und durften nicht einfach schlampig dargestellt werden. Für den Anfang sind wir der Meinung, das zwischen Musik, Bilder, Videos, Text, Programm, Archiv und Ordner unterscheidet werden soll, alles andere wird als Leer-File dargestellt[Abb.17].



[Abb.17]



Eventuell wurden auch die Icons überarbeitet, da diese zum Design des Emblems dazugehören müssen [Abb.18].



[Abb.18]