



TGM - HTBLuVA Wien XX  
IT Abteilung

---

# Diplomarbeit

## DigitalSchoolNotes

---

Version: 23. März 2016 um 19:05

# Inhalt

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Problembeschreibung</b>	<b>1</b>
2.1	Umfeldanalyse . . . . .	1
2.2	Projektidee . . . . .	2
2.3	Projektkoordination . . . . .	2
2.3.1	Kurzeinführung in Scrum . . . . .	2
2.3.2	Scrum im Team . . . . .	2
<b>3</b>	<b>Stand der Technik</b>	<b>3</b>
3.1	Frameworks . . . . .	3
3.2	Technologien . . . . .	4
<b>4</b>	<b>Design</b>	<b>5</b>
4.1	Software-Architektur . . . . .	5
4.2	Graphische Oberfläche . . . . .	6
4.3	Javascript Optimierung . . . . .	7
<b>5</b>	<b>Implementierung</b>	<b>8</b>
5.1	Infrastruktur und Testing . . . . .	8
5.1.1	Infrastruktur . . . . .	8
5.1.2	Testing . . . . .	14
5.2	User und Rollenmanagement . . . . .	18
5.2.1	Usermanagment . . . . .	18
5.2.2	Authentisierung . . . . .	18
5.2.3	Datenmodell . . . . .	20
5.2.4	Email . . . . .	21
5.2.5	Anmelden . . . . .	22
5.2.6	Usersicht . . . . .	23
5.2.7	Adminsicht . . . . .	24
5.3	Datenmanagement . . . . .	30
5.3.1	Persistierung von Daten . . . . .	30
5.3.2	Datenmodell . . . . .	30
5.3.3	Datenzugriff . . . . .	33
5.4	Parallel Working System . . . . .	35
5.5	Optical Character Recognition . . . . .	36
<b>6</b>	<b>Auswertung und Benchmarks</b>	<b>37</b>
<b>7</b>	<b>Ausblick</b>	<b>38</b>

<b>8</b>	<b>Zusammenfassung</b>	<b>39</b>
<b>9</b>	<b>Anhang</b>	<b>i</b>
9.1	Glossar . . . . .	i
9.2	Abbildungen . . . . .	ii
9.3	Listings . . . . .	ii
9.4	Quellen . . . . .	ii



# Abstract

Nowadays most people can't think of a world without internet anymore. It is used almost always and everywhere you are - also in schools. Students are getting more and more into digital notes, especially in technical oriented schools, where they have their laptop always with them. There is only one problem: These digital notes are often written in different programs. They are unorganized and mostly left in some directory where they aren't opened once afterwards.

The project DigitalSchoolNotes was formed exactly because of this problem. Our web-application can organize digital notes and make it easier to write them. It is made from students themselves to fit the needs of students optimal. The application can be accessed from almost every device - personal computers, laptops and even tablets. Furthermore it is possible to use a mobile phone to take a photo and put it into the digital notes easy and conveniently. Particularly fitted for technical schools there is also the possibility to insert some code-snippets into the digital notes, which are correctly formatted and highlighted.

All of which is to allow students to write tidy, organized, digital notes and to make learning from these easier and more efficient.

# Kurzfassung

Die heutige digitale Welt ist aus den Köpfen der meisten Menschen gar nicht mehr wegzudenken. Das Internet wird beinahe immer und überall verwendet - so auch immer mehr in Schulen. Vor allem in technischen Schulen werden Mitschriften aus dem Unterricht immer öfters digitalisiert. Das Problem dabei: Diese Mitschriften werden meist in unterschiedlichen Programmen verfasst, sie sind unorganisiert und verenden oft in irgendeinem Ordner ohne jemals wieder angesehen zu werden.

Das Projekt DigitalSchoolNotes setzt genau bei diesem Problem an. Unsere Web-Applikation soll das Führen einer digitalen Mitschrift einfacher und organisierter machen. Dabei wollen wir speziell auf die Bedürfnisse der Schüler eingehen. Der Zugriff auf die Applikation ist sowohl von Desktop Systemen und Laptops, als auch von Tablets über eine Webseite möglich. Ein eingeschränkter Zugriff ist zudem für Handys möglich, um beispielsweise Tafelbilder schnell und bequem in die Mitschrift einfügen zu können. Speziell für technisch orientierte Schulen gibt es auch die Möglichkeit, Teile von Programmcode richtig formatiert erstellen zu können.

Das alles dient dazu, Schülern eine ordentliche, organisierte, digitale Mitschrift zu erleichtern und das Lernen aus diesen effizienter zu gestalten.

# Danksagungen

# **1 Einleitung**

## **2 Problembeschreibung**

### **2.1 Umfeldanalyse**

## **2.2 Projektidee**

## **2.3 Projektkoordination**

### **2.3.1 Kurzeinführung in Scrum**

### **2.3.2 Scrum im Team**



## **3 Stand der Technik**

### **3.1 Frameworks**

## 3.2 Technologien

## 4 Design

### 4.1 Software-Architektur

## 4.2 Graphische Oberfläche

## 4.3 Javascript Optimierung

## 5 Implementierung

### 5.1 Infrastruktur und Testing

#### 5.1.1 Infrastruktur

Eine stabile und sichere Infrastruktur und gut getestete Software ist heutzutage ein Muss für jedes IT Projekt.

Die Infrastruktur ist wichtig, da in der Vergangenheit oft kleine Projekte bereits wenige Tage nach Veröffentlichung von sehr hohen Userzahlen berichten konnten. Wenn hier zuvor die Infrastruktur gut geplant und implementiert wurde, ist es kein Problem viele User zu bewältigen.

Ohne Tests wird heute keine Software mehr veröffentlicht, da etwaige Fehler für die Benutzer sehr abschreckend sein können bzw. dem Unternehmen viel Geld kosten können.

##### 5.1.1.1 Serverhosting

Die wichtigste technische Grundlage für das Projekt DigitalSchoolNotes ist der Projektserver. Auf diesem Server, wird das Projekt entwickelt und getestet. Hier ist es besonders wichtig, dass das gesamte Team mit der gleichen Umgebung arbeitet, da sonst die einzelnen Codeteile des Teams nicht zusammen funktionieren. Desweiteren wird der Server dazu verwendet, die Zwischenversionen des Projektes öffentlich verfügbar zu machen. Dies ist für das Team essentiell, da dadurch der Stakeholder jederzeit Zugriff auf eine aktuelle und stabile Version des Projektes hat. Dadurch kann das Team Änderungswünsche des Stakeholders leichter erfassen und realisieren.

Für die Auswahl des Serverhosters wurden einige Kriterien festgelegt. Diese lauten wie folgt:

- **Serverstandort:** Der Standort des Projektserver sollte möglichst nahe beim Endbenutzer sein, um die Latenz gering zu halten.
- **Verfügbarkeit:** Der Server sollte eine hohe Mindestverfügbarkeit haben. Dadurch kann sich der Endbenutzer darauf verlassen, dass das Service erreichbar ist. Der Minimalwert für die Verfügbarkeit wurde auf 99,6% festgelegt. Das bedeutet, dass der Server für maximal 35h im Jahr nicht verfügbar ist.
- **Support:** Der Hoster sollte Support unter der Woche und in Notfällen rund um die Uhr bieten.
- **Preis:** Um die Entwicklungskosten möglichst gering zu halten wurde der maximale Monatspreis auf 10 festgelegt.
- **Wartung:** Der Server sollte sich über ein Webinterface warten lassen.

Die oben genannten Kriterien reduzierten die Anzahl der möglichen Hoster stark. Das Team entschied sich für den Hoster netcup GmbH mit Sitz in Deutschland. Dieser erfüllte alle Anforderungen und Teile des Teams hatten bereits gute Erfahrungen mit dieser Firma gemacht.

Das ausgewählte Produkt der netcup GmbH heißt "Root-Server M v6". Dieser bietet folgende Features:

- **Virtualisierungstechnik:**KVM
- **CPU:**Intel®Xeon® E5-26xxV3 2,3GHz 2Cores
- **RAM:**6GB DDR4
- **Speicher:**120GB SSD

#### 5.1.1.2 Erreichbarkeit

Der Server ist unter der IP-Adresse 37.120.161.195 erreichbar. Da IP-Adressen schwer zu merken sind wurde ebenfalls eine Domain für das Projekt gekauft. Diese lautet "digitalschoolnotes.com" und löst auf die oben genannte IP-Adresse auf.

#### 5.1.1.3 Benutzerverwaltung am Projektserver

Jedes Projektteam Mitglied hat einen eigenen Unix Account auf dem Projektserver. Der Vorname der Person ist der Benutzername. Das Benutzerpasswort ist von jedem Teammitglied selbst gewählt. Alle Teammitglieder haben sudo rechte.

#### 5.1.1.4 Mailsystem

Das Projektteam hat einen Email-Verteiler mit der Adresse info@digitalschoolnotes.com. Jedes Teammitglied hat eine E-Mail Adresse nach dem Schema des TGMs(z.B. n hohenwarter@digitalschoolnotes.com).

Der Scrummaster ist unter scrummaster@digitalschoolnotes.com erreichbar.

#### 5.1.1.5 Serverzugriff

Um den Server zu konfigurieren und zu verwalten wird mit dem Protokoll SSH darauf zugegriffen. Aus Sicherheitsgründen wurde die Anmeldung mit Passwort verboten und es können hierfür nur noch SSH Keys verwendet werden. Diese sind um einiges sicherer.

### 5.1.1.6 Firewall

Um den Server vor Angriffen und unerwünschten Zugriffen zu schützen wurde eine Firewall installiert. Diese blockiert alle unerwünschten Anfragen. Prinzipiell sind alle Ports geschlossen. Es werden nur Ports geöffnet, welche für das Betreiben des Projektes notwendig sind.

Es folgt eine Liste der freigegebenen Ports:

- 22 SSH
- 53 DNS
- 80 HTTP
- 443 HTTPS
- 5001-5005 Django Development

Die Konfiguration der Firewall des Projektserver sieht wie folgt aus:

```
# Flush the tables to apply changes
iptables -F

# Default policy to drop 'everything' but our output to internet
iptables -P FORWARD DROP
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT

# Allow established connections (the responses to our outgoing traffic)
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Allow local programs that use loopback (Unix sockets)
iptables -A INPUT -s 127.0.0.0/8 -d 127.0.0.0/8 -i lo -j ACCEPT
iptables -A FORWARD -s 127.0.0.0/8 -d 127.0.0.0/8 -i lo -j ACCEPT

#Allowed Ports
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 53 -m state --state NEW -j ACCEPT
iptables -A INPUT -p udp --dport 53 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 5001 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 5002 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 5003 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 5004 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 5005 -m state --state NEW -j ACCEPT
```

Listing 1: Firewall Rules des Projektserver



Da normalerweise nach einem Reboot des Servers die Firewallkonfiguration verloren geht, musste diese persistiert werden. Das wird durch das Paket *iptables-persistent* erledigt. Die Konfiguration dieses Paketes geschieht wie folgt[Kre15b]:

```
# Install
sudo apt-get install iptables-persistent

# Save Rules
iptables-save > /etc/iptables/rules.v4
```

Listing 2: Installation iptables-persistent

#### 5.1.1.7 Bruteforce Prevention

Um Bruteforce Angriffe auf den SSH Dienst zu erschweren wurde am Server fail2ban eingerichtet. Dieses Tool zählt fehlgeschlagene Anmeldeversuche mit und sperrt die IP Adresse des Angreifers nach einer festgelegten Anzahl an Versuchen. Dieses Verfahren ist äußerst effektiv, da der Angreifer dadurch keine Chance hat eine große Anzahl an Passwörtern auszuprobieren(z.B. Wörterbuchangriff). Da auf dem Projektserver die Anmeldung nur mit SSH Key möglich ist, hat der Client welcher sich zum Server verbinden will sechs Versuche einen korrekten SSH Key zu übermitteln.

#### 5.1.1.8 Webserver

Als Webserver für unsere Applikation wurde Nginx gewählt. Dieser wurde vor allem gewählt, da das Team bereits in der Vergangenheit mit dieser Software gearbeitet hat. Mithilfe von Nginx kann ebenfalls ein Loadbalancer realisiert werden. Dies ist ein wichtiger Punkt um die Software skalierbar zu halten.

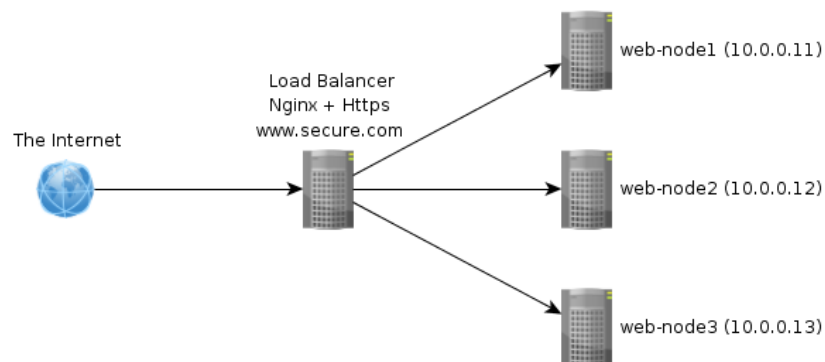


Abbildung 1: Loadbalancing

Der Webserver ist hauptsächlich für den statischen Content(HTML, Javascript, CSS, Bilder...) zuständig. Die funktioniert indem alle statischen Inhalte in einem Ordner abgelegt werden. Damit weiß Nginx, dass er für diese Inhalte zuständig ist.

#### **5.1.1.9 SSL**

Um die Daten und Privatsphäre unserer Kunden zu schützen wird bei allen Aufrufen der Website mit SSL verschlüsselt. Um eine legitime SSL Verschlüsselung zu gewährleisten ist ein valides Zertifikat notwendig. Dieses muss von einer Zertifizierungsstelle erworben werden. Das verwendete Zertifikat für das Projekt wurde von der Zertifizierungsstelle namens thawte Inc. erworben.

Das Zertifikat validiert die Domain(Domain Validated). Das bedeutet, dass zur Ausstellung des Zertifikates eine Email an den Besitzer der Domain geschickt wird. Wenn der Besitzer der Domain der Zertifizierung zustimmt wird diese durchgeführt.

Um das Zertifikat nun verwenden zu können muss es mit dem Intermediate Zertifikat der Zertifizierungsstelle verbunden werden. Dadurch ist ein eindeutiger Zertifizierungsfluss hergestellt. Dannach kann es auf den Webserver deployed werden.

#### **5.1.1.10 Produktivbetrieb**

Im Produktivbetrieb ist der Betrieb der Software auf zwei Dienste aufgeteilt. Der statische Teil der Applikation wird wie bereits vorhin beschrieben von Nginx dem User zur Verfügung gestellt.

Der dynamische Teil - das Backend - stellt unser Django Server dar. Hier werden die kritischen Operationen wie Datenbankzugriffe oder die Authentifizierung durchgeführt. Wird nun unsere API (der dynamische Teil) aufgerufen, leitet Nginx die Anfrage an den Django Server weiter.

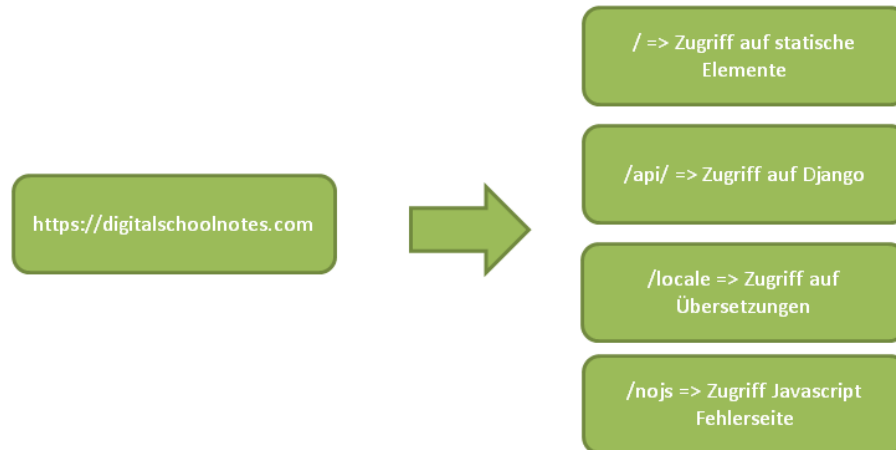


Abbildung 2: Deployment Struktur

Der Django Server läuft mittels gunicorn. Gunicorn ist ein WSGI HTTP Server und somit die Schnittstelle zwischen dem Webserver und Django. Gunicorn startet für Django mehrere Worker Prozesse, wodurch die Anfragen theoretisch auf mehrere CPU Kerne aufgeteilt werden können. Desweiteren startet es die einzelnen Prozesse automatisch neu falls diese abstürzen.

#### 5.1.1.11 Testbetrieb

Der Testbetrieb läuft relativ ähnlich wie der Produktivbetrieb ab. Jedes Teammitglied arbeitet an einer eigenen Instanz des Codes. Dadurch behindert sich das Team nicht gegenseitig falls Fehler auftreten. Um dies zu ermöglichen hat jede Person einen eigenen Port zugewiesen bekommen auf der seine Version der Applikation zu erreichen ist.

Hat diese Person nun eine Änderung am Code vorgenommen muss dieser auf den Server hochgeladen werden. Nun kann innerhalb der IDE der Django Server gestartet und beendet werden. Dies ist wichtig, da dadurch auch die Fehlermeldungen von Django innerhalb der IDE sichtbar sind. Für den statischen Teil ist auch hier Nginx zuständig.

#### **5.1.1.12 Verfügbarkeit**

Um in Zukunft die Verfügbarkeit zu verbessern gibt es viele Möglichkeiten. Um die Wahrscheinlichkeit eines kritischen Serverausfalles zu reduzieren könnten mehrere Server angemietet werden. Eine andere Möglichkeit wäre es das Hosting in eine Cloud auszulagern (Amazon Web Services, Google, Azure...).

Um die Performance und Verfügbarkeit zu erhöhen sollte ein Loadbalancer verwendet werden. Dieser teilt die eingehenden Anfragen auf mehrere andere Server auf und reduziert damit die Last der einzelnen Server und erhöht die Performance. Hierbei ist darauf zu achten, dass der Loadbalancer Healthchecks durchführen muss. Mithilfe dieser kann der Loadbalancer überprüfen ob die Server noch aktiv sind und somit Anfragen entgegennehmen und bearbeiten können. Sollte ein Server ausfallen leitet der Loadbalancer keine Anfragen mehr an diesen weiter. Der Loadbalancer sollte redundant ausgeführt sein, da ohne ihn die Seite nicht mehr erreichbar ist.

Um die Performance der Datenbank zu erhöhen sollte diese auf einem eigenen Server betrieben werden. Des Weiteren sollte hier Loadbalancing oder Clustering betrieben werden um die Ausfallssicherheit zu erhöhen.

Die Applikation sollte in Containern (z.B. Docker) mit Containermanagement deployed werden um die Verfügbarkeit zu erhöhen. Somit könnte bei einem Fehler der Container einfach entfernt und ein neuer gestartet werden.

Um die Erreichbarkeit der Domain zu gewährleisten sollten redundante DNS Server verwendet werden. Hier gibt es auch die Möglichkeit ein CDN Network (z.B. Cloudflare, Cloudfront...) zu verwenden um einen Performancevorteil zu erlangen. Durch das CDN wird auch die Serverlast reduziert.

### **5.1.2 Testing**

#### **5.1.2.1 Allgemeines**

Um eine Einwandfreie Benutzung einer Software zu ermöglichen muss diese getestet werden. Mittels Tests können Fehler gefunden und repariert werden. Um Zeit zu sparen werden diese Tests mit Testing-Frameworks automatisiert.

### 5.1.2.2 Testing Level

Beim Softwaretesting wird in zwei Kategorien eingeteilt: Functional Testing und Non-functional Testing. Functional Testing testet definierte Spezifikationen. Damit soll sichergestellt werden, dass die Applikation sich so verhält wie erwartet. Non-functional Testing testet Dinge wie Performance und Erreichbarkeit.[tut15]

Das Functional Testing lässt sich in vier Level aufteilen, welche auch in folgender Grafik dargestellt werden.

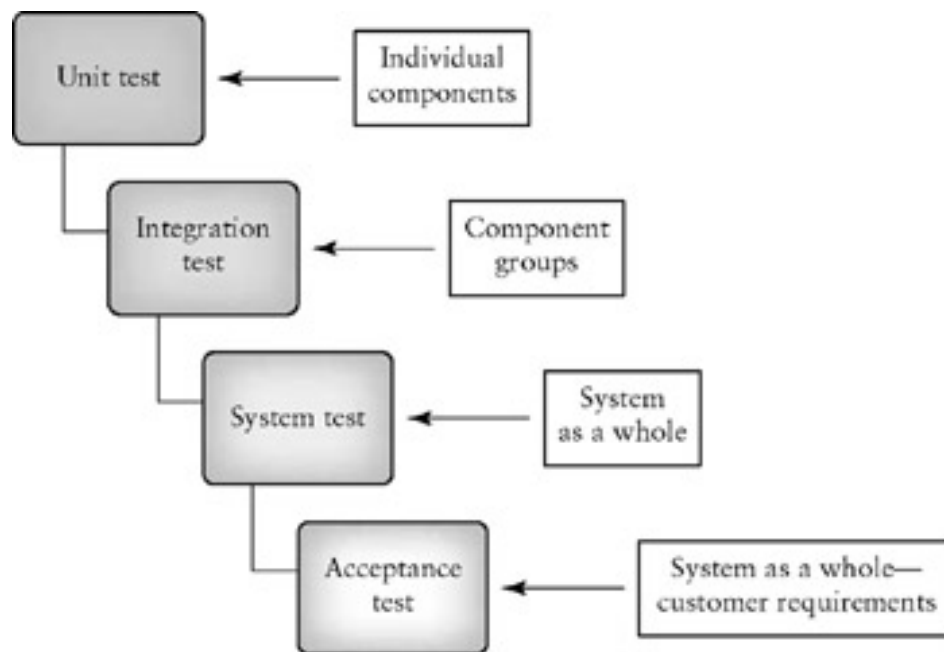


Abbildung 3: Functional Testing Level

Mithilfe von Unit Tests wird die Funktionalität des Programmes überprüft. Hier werden Methoden oder Programmteilen - sogenannten Units - Daten übergeben. Diese werden dann verarbeitet und das Ergebnis wird überprüft. Diese Tests werden bereits während der Entwicklung erstellt und stellen sicher, dass neue Änderungen am Code den bereits vorhandenen Code nicht kompromitieren. [Pea15]

Integration Tests prüfen die Kompatibilität zwischen dein einzelnen Units. Es kann sein, dass Units alleine funktionieren, jedoch zusammen mit anderen Units nicht. Des Weiteren wird hier auch die Performance überprüft. Jede einzelne Unit kann effizient sein, sind die Units jedoch schlecht integriert entsteht trotzdem ein ineffizientes Programm. [Pea15]

Bei System Tests wird die ganze Applikation getestet. Hier ist es das Ziel die Software auf Erfüllung der Qualitätsstandards und Anforderungen zu Testen. Diese Tests werden von eigenständigen Testern durchgeführt, welche nicht an der Entwicklung des Programmes beteiligt waren. [Pea15]

Acceptance Tests sind das letzte Test Level. Hier wird geprüft ob die Softwareversion für eine Veröffentlichung bereit ist. Dazu müssen die Endanwender ausprobieren ob die Software wie gewünscht funktioniert. [Pea15]

### 5.1.2.3 Testerstellung

Die einzelnen Tests werden in Java programmiert. Beim ausführen der Tests wird ein Browser gestartet welcher dann die gewünschten Websiteaufrufe tätigt. Tests können zu Test Suites zusammengefasst werden. Dadurch werden alle Tests in der Suite nacheinander ausgeführt.

Um im Browser auf ein Element mit Selenium klicken zu können muss dieses eindeutig identifizierbar sein. Hierbei gibt es mehrere Möglichkeiten. Eine eindeutige Identifizierung ist über ID, Name, CSS Klasse, Link Text oder XPATH möglich.

```
driver.findElement(By.id("email")).click();
driver.findElement(By.name("email")).clear();
driver.findElement(By.className("cssclass")).click();
driver.findElement(By.partialLinkText("Klick mich")).click();
driver.findElement(By.xpath("//*[contains(text(), 'Mail senden')]")).click();
```

Listing 3: Selenium Element Selektoren

Selenium kann auch Formulare ausfüllen. Dazu muss das Element eindeutig angesprochen werden und dann kann wie oben im Beispiel statt dem *.click()* ein *.sendKeys()* verwendet werden.

Ein kompletter Test sieht dann wie folgt aus:

```
public void testInvalidUser() throws Exception {
    driver.get(baseUrl + "/login");
    driver.findElement(By.name("email")).clear();
    driver.findElement(By.name("email")).sendKeys("testinactive@test.test");
    driver.findElement(By.name("pwd")).sendKeys("12341234");
    driver.findElement(By.id("submit")).click();
    Thread.sleep(Parameters.SLEEP_PAGELOAD);
    String page = driver.getPageSource();
    if(!page.contains("Bitte bestaetige zuerst deine E-Mail Adresse!")) throw
        new NotFoundException();
}
```

Listing 4: Selenium Test

#### 5.1.2.4 Probleme

Die Verwendung des Selenium Frameworks bereitete einige Probleme. Selenium kann nur mit Elementen interagieren welche auch sichtbar sind. Dadurch können z.B. Buttons welche nur bei Hover angezeigt werden schwer getestet werden. Hierfür muss Selenium zuerst die Maus in die Hover Region bewegen und kann erst dann den Button drücken

Um ein Element Testen zu können muss es eindeutig identifizierbar sein. Dies ist allerdings bei vielen Elementen in unserer Software nicht möglich. Zum Beispiel sind die einzelnen Buttons im Texteditor im Heft nicht eindeutig identifizierbar. Diese können also nicht automatisiert getestet werden. Stattdessen wurden hier geführte Acceptance Tests durchgeführt. Das bedeutet, dass Selenium die Seite aufruft, eine Anweisung an den User gibt und dannach fragt ob das gewünschte Ergebnis eingetroffen ist.

Die auf DSN verwendeten Captchas bei der Registrierung und dem Passwort Reset können ebenfalls nicht automatisiert getestet werden. Captchas sollen im Endeffekt die automatische Ausfüllung eines Formulars ja auch verhindern. Bei einem Test füllt Selenium das Formular bis auf das Captcha aus. Das Captcha muss dann händisch gelöst werden.

Die Tests werden relativ langsam ausgeführt. So dauert es ca. 30 Minuten alle DSN Tests auszuführen. Dies liegt vor allem an der Netwerklatenz der einzelnen Websiteaufrufe. Die Tests könnten parallelisiert auf mehreren Servern ausgeführt werden, jedoch ist die Einrichtung eines solchen Testing Clusters kompliziert.

## 5.2 User und Rollenmanagement

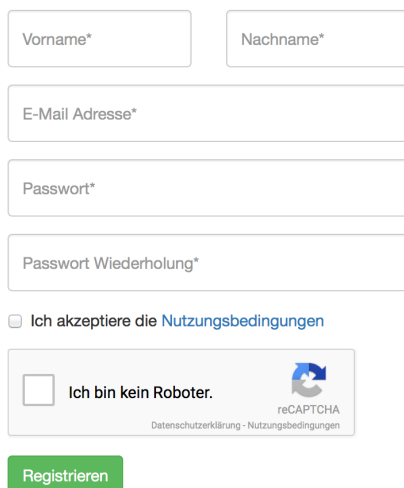
### 5.2.1 Usermanagement

Unter dem Begriff Usermanagement versteht sich, dass verwalten und kontrollieren von Benutzerkonten. Es soll dazu dienen jeden registrierten User eindeutig zu identifizieren und zu kontrollieren, ob die monatlichen Raten für einen Pro-Account überwiesen wurden. Außerdem soll die bereits in Anspruch genommene Speicherkapazität überwacht werden. Dazu ist notwendig, dass jeder User durch eine Kombination von Daten, einmalig, unterscheidbar von anderen ist.

### 5.2.2 Authentisierung

Unter Authentisierung versteht man den Nachweis der behaupteten Identität der BenutzerInnen. Im Falle von DSN handelt es sich hierbei um die eindeutige Email-Adresse, welche einmalig im System benutzt wird. Unter der Identität versteht sich die Sicherheit von wem die Information stammt. Jedes Handeln eines Benutzers kann jemanden zugewiesen werden.

Ein weiterer Identitätspunkt wäre, dass zu geheim haltenden Passwort, welches aus Sicherheitsgründen mindestens 8 Zeichen beinhalten muss. Durch 8 Zeichen möchten wir Cyberkriminelle das Knacken von Passwörtern erschweren. Für die Abschließung der Registrierung müssen die Nutzungsbedingungen akzeptiert werden. "Allgemeine Geschäftsbedingungen (AGB) sind vertragliche Klauseln, die zur Standardisierung und Konkretisierung von Massenverträgen dienen. Sie werden von einer Vertragspartei einseitig gestellt und bedürfen daher einer bes. Kontrolle, um ihren Missbrauch zu verhindern." [DCH15] [AST08][Kre15a]



The image shows a registration form for DSN. It consists of several input fields: 'Vorname\*' (First Name), 'Nachname\*' (Last Name), 'E-Mail Adresse\*' (Email Address), 'Passwort\*' (Password), and 'Passwort Wiederholung\*' (Password Repeat). Below these fields is a checkbox labeled 'Ich akzeptiere die Nutzungsbedingungen' (I accept the terms of use). Further down is a reCAPTCHA widget with the text 'Ich bin kein Roboter.' (I am not a robot.) and a small robot icon. At the bottom of the form is a green button labeled 'Registrieren' (Register).

Abbildung 4: Authentisierung bei DSN



Um auszuschließen, dass sich eine Software bzw. ein Roboter einen Account auf DSN erstellt, wird ein Captcha verwendet. Ein Captcha dient zur Sicherheit und soll überprüfen wer die Eingabe tätigte. Um das Captcha anzuzeigen muss eine JS Lib eingebunden werden [Sha15].

```
<script src="https://www.google.com/recaptcha/api.js?
onload=vcRecaptchaApiLoaded&render=explicit" async defer></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular-recaptcha/2.2.5/
angular-recaptcha.min.js"></script>
```

Listing 5: Einbindung der JS-Library Recaptcha

Dannach kann das Captcha einfach wie folgt ins Form eingebunden werden:

```
<div vc-recaptcha key="publicKey"></div>
```

publicKey definiert den Public Key des Recaptchas. Dieser wird im Javascript file über \$scope zugewiesen. Das Captcha wird nach der Benutzereingabe serverseitig validiert. Mit den Daten des Registrierungsformular wird ein weiteres Attribut namens recaptcha erhalten. In diesem steht ein Key der zur Validierung an Google gesendet werden muss. Google möchte zur Validierung den Key, die IP des Users und den Secret App Key. Google gibt dann zurück, ob alles korrekt ablief, also ob es sich bei dem User tatsächlich um ein menschliches Lebewesen handelt. Um die Validierung mehrmals einsetzen zu können, haben wir eine Methode dafür geschrieben:

```
def validate_captcha(recaptcha, ip):
    response = {}
    url = "https://www.google.com/recaptcha/api/siteverify"
    params = {
        'secret': settings.RECAPTCHA_SECRET_KEY,
        'response': recaptcha,
        'remoteip': ip
    }
    verify = requests.get(url, params=params, verify=True)
    verify = verify.json()
    response["status"] = verify.get("success", False)
    if response["status"] == True:
        return True
    else:
        return "Captcha ist nicht valide."
```

Wenn das Captcha gelöst wurde, kann man damit das Form genau ein Mal absenden. Im Falle, dass das Formular nochmals abgesendet werden muss, ist es notwendig das Captcha zurück zu setzen.

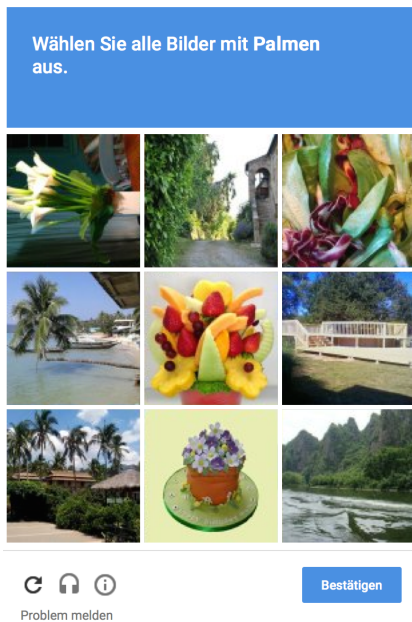


Abbildung 5: Auslösen des Captchas

### 5.2.3 Datenmodell

Da Django-Authentifizierungs Funktionalitäten nur für relative DBMS ausgelegt sind, daher für den vorgesehenen Anwendungszweck nicht geeignet sind, mussten Änderungen vorgenommen werden, um die Authentifizierung über MongoDB zu ermöglichen.

Nun muss das User-Model, das eben auf Mongo Basis definiert wurde, noch im File models.py erstellt werden. Der Code für das Model wurde aus dem entsprechenden Source-Code von MongoEngine [Git15] kopiert und an unseren Anwendungszweck angepasst.

```
class User(Document):
    id = ObjectIdField(unique=True, required=True, primary_key=True)
    email = EmailField(unique=True, required=True)
    first_name = StringField(max_length=30)
    last_name = StringField(max_length=30)
    password = StringField(max_length=128)
    is_staff = BooleanField(default=False)
    is_prouser = BooleanField(default=False)
    is_active = BooleanField(default=True)
    is_superuser = BooleanField(default=False)
    last_login = DateTimeField(default=datetime.datetime.now())
    date_joined = DateTimeField(default=datetime.datetime.now())
    passwordreset= EmbeddedDocumentField>PasswordReset)
    user_permissions = ListField(ReferenceField(Permission))
    [...]
```

Da MongoDB den PrimaryKey als ObjectId verlangt, mussten im bestehenden Original Django-Code folgende Änderungen vorgenommen werden [Kha15]:

Im File `usr/local/lib/python3.4/dist-packages/django/db/models/fields` in Zeile 964: `return int(value)` ändern zu: `return value` Im File `/usr/local/lib/python3.4/dist-packages/django/contrib/auth/models.py` in Zeile 111: `request.session[SESSION_KEY] = user._meta.pk.value_to_string(user)` ändern zu: `try: request.session[SESSION_KEY] = user._meta.pk.value_to_string(user) except Exception: request.session[SESSION_KEY] = user.id`

Nach der Authentisierung wird der/die zukünftige BenutzerIn nach dem beschriebenen Datenmodell in die MongoDB hinzugefügt. Da der Registrierungsprozess noch nicht abgeschlossen ist, ist das `isactive` Attribut noch auf `False` gesetzt. Solange das Attribut nicht in einen anderen Zustand übergeht, ist dem/der BenutzerIn nicht erlaubt sich auf DSN anzumelden. Es kann auch sein, dass sich ein bereits registrierter Account sich in diesem Dilemma befinden kann. Das könnte z.B. der Fall sein wenn jemand gegen die Regeln und Gesetze von DSN verstößt und daraufhin gesperrt wird. Dann bleibt ihm nichts anderes übrig Kontakt mit dem Administrator aufzunehmen. DATENBANK AUFRUF MAXMUSTERMANN

#### 5.2.4 Email

Um den Registrierungsprozess zu beenden, wird dem nahestehenden User ein Token per Mail zugesendet. Dieser dient der Identifizierung und Authentifizierung und könnte folgendermaßen aussehen `http://digitalschoolnotes.com/validate/dad9574635aad7d6549536db38f7839c042f7704b3bd74acc427f075d0601470`. Bei der Erstellung eines solchen Tokens werden die Email-Adresse des Benutzer, welche als Nachweis dient um zu wissen welcher Account aktiviert werden soll und das aktuelle Datum kombiniert. Diese werden miteinander verknüpft, in einen Hash umgewandelt und in die Datenbank abgespeichert. *"validatetoken"*:

*"f043ea6e44aea716d08ae2cb70d91bcb50196da1eb89b4727c124508dbf0d85"*

Das Datum dient dazu um dem Token ein Zeitstempel zu geben, dieser bezweckt die Gültigkeit. Wenn dieser Hash nicht in den kommenden 24 Stunden eingelöst wird, verfällt dieser Token, da der Zeitstempel abgelaufen ist und es muss vom User ein neuer angefordert werden. Im File `authentication/registration.py` in Zeile 23 ist die Umsetzung des Registrierungstoken dargestellt:

```
def create_validation_token(email):
    user = User.objects.get(email=email)
    now = datetime.datetime.now()
    to_hash = (str(user.id) + str(now)).encode('utf-8')
    hashed = hashlib.sha256(to_hash).hexdigest()
    hashed = str(hashed)
    user.validatetoken = hashed
    user.save()
    return 'http://digitalschoolnotes.com/validate/' + hashed
```

Um eine Email verschicken zu können müssen folgende Konfigurationen unternommen werden. Im *settings.py* muss der Email-Server definiert werden.

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'mx92d.netcup.net'
EMAIL_HOST_USER = 'noreplyATdigitalschoolnotes.com'
EMAIL_HOST_PASSWORD = 'passwort'
EMAIL_PORT = 25
EMAIL_USE_TLS = False
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```


### 5.2.5 Anmelden

Wenn der neue Benutzer den Registrierprozess erfolgreich abgeschlossen hat, steht ihm/ihr jetzt frei sich anzumelden. Entweder durch Angabe der Email-Adresse und Passwort oder mittels OAuth. Mit OAuth besteht die Möglichkeit die Registrierung auszulassen und sich direkt anzumelden. OAuth steht für Open Authentication und bietet dem Nutzer die Möglichkeit Daten über einen Webserverice auszutauschen. „OAuth sichert die Programmschnittstelle von Web-Anwendungen ab und verwendet für die Übertragung der Nutzeridentifikation dessen Passwort und einen Token“[DI15]. Bei dem Zugriff auf die sensiblen Daten muss der Benutzer keine zusätzlichen Information und auch keine Identität preisgeben. Der Provider holt sich die Benutzerdaten von Facebook oder Google+ und erstellt für den User einen Account.

**Anmelden**

[Passwort vergessen?](#)

Durch die Anmeldung mit Facebook oder Google+ akzeptierst du unsere [Nutzungsbedingungen](#)






Abbildung 6: Klassische Anmeldung oder mittels OAuth

Um einen Benutzer anzumelden, muss zunächst ein User-Objekt mit der übergebenen Email Adresse von der Datenbank abgefragt werden. Sollte diese E-Mail Adresse keinem Benutzer zugeordnet sein, existiert der Benutzer noch nicht. Ansonsten muss mit *.check\_password(...)* das eingegebene Passwort überprüft werden. Sollte dieses korrekt

sein, kann der User angemeldet werden. Dazu muss das Authentication-Backend und ein Session Timeout gesetzt werden und der Benutzer über die Funktion *login()* eingeloggt werden:

```
try:
    user = User.objects.get(email='exampleATexample.com')
except:
    user = None
if user is not None and user.check_password('myPassword'):
    user.backend = 'mongoengine.django.auth.MongoEngineBackend'
    login(request, user)
    request.session.set_expiry(60 * 60 * 1) # 1 hour timeout
```

Der aktuell angemeldete Benutzer kann mittels *request.user* abgefragt werden. Sollte der Benutzer aktuell nicht angemeldet sein, ist dies null, ansonsten wird das entsprechende User-Objekt zurückgeliefert. Um einen angemeldeten Benutzer wieder abzumelden muss lediglich folgende Funktion ausgeführt werden: *logout(request)*

### 5.2.6 Usersicht



Abbildung 7: Navigationsleiste als User

Ein angemeldeter Benutzer kann seine Benutzerinformationen nachgiebig unter Kontoeinstellungen ändern. Für die Änderung seiner Daten, muss aus Sicherheitsgründen, dass aktuelle Passwort eingegeben werden.

Im System befinden sich drei verschiedene Berechtigungsstufen, welche sind: der Standard-Benutzer, Pro-Benutzer und Administrator. Jedem/r registrierten AnwenderIn ist zu Beginn ein Standard-Benutzer. Ihnen stehen eine begrenzte Anzahl an digitalen Hefen zur Verfügung. Durch eine geringe monatliche Zahlung kann der Standard-Account zum Pro-Account upgegradet werden, wodurch dem/r SchülerIn erweiterte Funktion angeboten werden. Zum einen stehen mehr Hefte zur Verfügung, es wird keine Werbung angezeigt, sowie keine Speicherbeschränkung. Die letzte Berechtigungsstufe sind Administratoren. Sie sind ebenfalls Pro-User, haben im Gegensatz einen eigenen Admin-Bereich, wo sämtliche Daten über Benutzer verwaltet und kontrolliert werden können. Dieser Bereich kann mit */admin* nach der URL aufgerufen werden. Er unterscheidet sich durch den schwarzen Menübalken.

Jede/r BenutzerIn hat die Möglichkeit nach seinen Freunden oder anderen registrierten Anwendern zu suchen. Rechts oben in der Navigationleiste befindet sich ein Suchbalken, der es Anwendern ermöglicht nach Vornamen, Nachnamen oder nach Email-Adressen zu suchen.

## Kontoeinstellungen

Vorname

Max

Nachname

Mustermann

E-Mail Adresse

max@mustermann.com

Aktuelles Passwort\*

Aktuelles Passwort

Passwort ändern

Neues Passwort

Neues Passwort

Abbildung 8: Kontoeinstellungen

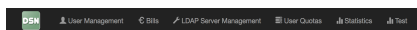
16 User wurden gefunden

test@test.test  
Test\_firstname Test\_lastname

Abbildung 9: Usersuche

Bei Klick auf den gefundenen User gelangt auf dessen Profil. Auf dem Profil ist zu einem der volle Name zu sehen, die Email-Adresse und die Berechtigungsstufe, ob es sich um einen Standardbenutzer, Pro-Benutzer oder Administrator handelt. Ein weiterer wichtiger Punkt sind die öffentlichen Hefte. Jeder User ist in der Lage von öffentlichen Heften andere einzelne Seiten zu exportieren.

### 5.2.7 Adminsicht



Auf der User Management Page werden alle Benutzer von DSN aufgelistet. Man hat Einsicht auf die Email-Adresse, Vorname, Nachname und auf die Berechtigungsstufe, Standard-Benutzer, Pro-Benutzer, Administrator. Außerdem besteht die Möglichkeit als Administrator andere Benutzer zu löschen, die Berechtigungsstufe zu ändern oder den Benutzer mittels einer Mail auf etwas hinzuweisen. Neben der Auflistung der Benutzer, kann auch nach einer bestimmten Person suchen. Die Eingabe wird mit den Vorname, Nachnamen und der Email-Adresse verglichen. Mittels `collectionName.objects()` liefert Mongoengine alle Objekte von der angegeben Collection. Wenn der Administrator

aber nicht alle Objekte von einer Collection haben möchte, sondern nur eine gewisse Anzahl, können diese mit folgenden Befehl abgefragt werden: *Tabellenname.objects[x:y]*  
Objekte können wie folgt gesucht werden:

*users(Q(email\_\_icontains=suchtext) | Q(first\_name\_\_icontains=suchtext) | Q(last\_name\_\_icontains=suchtext))*

Objekte können auch nach einer bestimmten Spalte sortiert werden.

*users.order\_by(spaltenname) users.order\_by('-'+spaltenname)*

Falls man ein vorhandenes Objekt aus der Collection löschen möchte, muss dieses zuvor rausfiltern und dann mit der Funktion *delete()* entfernen. Veränderte Daten werden mit der Funktion *save()* persistiert.

#### Usermanagement

Search:

Email-Adresse *	Vorname	Nachname	Berechtigungsstufe	Löschen	E-Mail
z@z.com	z	z	Inaktiv	Account löschen	Mail senden
y@yy.vom	y	y	Inaktiv	Account löschen	Mail senden
xx@x.com	x	x	Inaktiv	Account löschen	Mail senden
vorname929@nachname929.test	Vorname929	Nachname929	Benutzer	Account löschen	Mail senden
vorname650@nachname650.test	Vorname650	Nachname650	Benutzer	Account löschen	Mail senden

Da unser User Management Page nur Seitenweise die Benutzer liefert, um Ladezeiten zu minimieren, muss diese mittels Pagination umgesetzt werden. Für die Realisierung sind die Anzahl der Objekte die insgesamt ausgegeben werden sollen gefordert. Dann wird definiert, wieviele Elemente pro Seite angezeigt werden sollen. Um die Seitenanzahl zu berechnen, werden alle Elemente durch die Anzahl der Elemente dividiert, die pro Seite dargestellt werden sollen. Der Server übergibt dann die Elemente, welche auf einer Seite angezeigt werden sollen. Falls der User auf eine andere Seite wechselt, werden die nächsten Objekte vom Server bezogen.

```
$http({
  method: 'GET',
  url: '/api/admin_user',
  data: {}
})
.success(function (data) {
  $scope.users = data['test'];
  $scope.len = data['len'];
  $scope.currentPage = 0;
  $scope.l = Math.ceil($scope.len/$scope.itemsPerPage);
})
.error(function (data) {
});
var searchMatch = function (haystack, needle) {
  if (!needle) {
```

```

        return true;
    }
    return haystack.toLowerCase().indexOf(needle.toLowerCase()) !== -1;
};

$scope.range = function (size, start, end) {
    var ret = [];
    if (size < end) {
        end = size;
        start = size;
    }
    for (var i = start; i < end; i++) {
        ret.push(i);
    }
    return ret;
};

$scope.firstPage = function () {
    $scope.currentPage = 0;
};

$scope.prevPage = function () {
    if ($scope.currentPage > 0) {
        $scope.currentPage--;
    }
};

$scope.nextPage = function () {
    if ($scope.currentPage < $scope.l- 1) {
        $scope.currentPage++;
    }
};

$scope.lastPage = function () {
    $scope.currentPage = $scope.l-1;
};

$scope.setPage = function () {
    $scope.currentPage = this.n;
};

```

```

def view_users(request):
    if not request.user.is_authenticated() or not request.user.is_superuser:
        return JsonResponse({})
    u = []

```



```

length = 0
weiter = False
delete = False
if request.method == "GET":
    users = User.objects[0:20]
    length = len(User.objects)
elif request.method == "POST":
    params = json.loads(request.body.decode('utf-8'))
    von = (params['Page']-1)*params['counter']
    bis = params['counter']*params['Page']

    try:
        """ Delete """
        user = User.objects.get(email=params['email'])

        if user != request.user and user.delete_date == None:
            enddate = datetime.now() + timedelta(days=7)
            until = date(enddate.year, enddate.month, enddate.day)
            user.delete_date = until
            user.save()
            deleteemail(user.email, user.first_name, until)
        elif user != request.user:
            user.delete_date = None
            user.save()
    except KeyError:
        pass

    users = User.objects()

    try:
        """ Search """
        if bool(params['text'] and params['text'].strip()):
            users = users(Q(email__icontains=params['text']) |
                          Q(first_name__icontains=params['text']) |
                          Q(last_name__icontains=params['text']))
    except KeyError:
        pass

    try:
        """ Sort """
        if params['order'] is not None:
            if params['order']:
                users = users.order_by(params['spalte'])
            else:
                users = users.order_by('-'+str(params['spalte']))
    except KeyError:
        pass

```

```

length = len(users)
users = users[von:bis]
for user in users:
    security = 1
    if user.is_prouser: security = 2
    if user.is_superuser: security = 3
    if not user.is_active: security = 4
    if user.delete_date == None:
        delete_state = 'Account loeschen'
    else:
        days = abs(datetime.today().day -
                    int(date.strftime(user.delete_date, "%d")))
        delete_state = ' Loeschung in %s Tagen' % (str(days))

    u.append({
        "email": user.email,
        "first_name": user.first_name,
        "last_name": user.last_name,
        "security_level": security,
        "delete_account": delete_state
    })
if length == 0:
    return JsonResponse({'test': u})
else:
    return JsonResponse({'test': u, 'len': length})

```

Falls ein Pro-User seinen Zahlungen nicht nachkommt oder sich durch Böswilligkeiten bemerkbar macht hat der Administrator von DSN das Recht diesen User zu löschen. DSN gibt den User die Möglichkeit seine Daten bzw. Hefte bevor er gelöscht wird zu sichern. Der zu löschende Benutzer empfängt eine Email, wo darauf hingewiesen wird das sein Account und alle dazugehörigen Daten nach 7 Tagen gelöscht werden. Am Server von DSN läuft ein cronjob welcher jeden Tag das definierte Command, welches überprüft wann der zu löschende User entfernt werden soll, ausführt. Im Falle das jemand länger als 3 Monate interaktiv ist, wird ihm eine Informationsmail zugesendet. Diese informiert ihn, dass er sich in den 7 kommenden Tagen einloggen soll, ansonsten wird der Account mit alle den Daten gelöscht. [Fou15][ubu15]

```

from django.core.management import BaseCommand
from datetime import *
from dsn.models import User
from dsn.authentication.account_delete import delete_account

#https://docs.djangoproject.com/en/1.9/howto/custom-management-commands/
#The class must be named Command, and subclass BaseCommand
class Command(BaseCommand):

```

```

# Show this when the user types help
help = "Command for the User notification"

# A command must define handle()
def handle(self, *args, **options):
    until = datetime.now() + timedelta(days=7)
    users = User.objects(delete_date__lte=until)
    for user in users:
        now = datetime.today()
        day = abs(now.day - int(date.strftime(user.delete_date, "%d")))
        if day == 0: #User delete
            delete_account(user)

```

## Cronjob

```

# m h dom mon dow  command
# * * * */1 * *    python3 /home/stable/dsn/manage.py inform
# * * * */1 * *    python3 /home/stable/dsn/manage.py delete

```

Unter dem Navigationspunkt Bills werden die Rechnungen von den Pro-Benutzern aufgelistet. Zum einen wann und ob der Betrag eingezahlt wurde.

## 5.3 Datenmanagement

Ein gutes Datenmanagement ist eine Grundvoraussetzung für eine gut funktionierende Applikation. Alle Daten müssen persistiert werden, um nicht verloren zu gehen. Zudem soll durch ein gut organisiertes Datenmanagement eine einfache und effiziente Verwaltung der Daten gesichert werden.

### 5.3.1 Persistierung von Daten

Die Persistierung von Daten ist besonders wichtig. Das Abspeichern aller Daten dient dem Zweck, die Daten auch zu einem späteren Zeitpunkt oder beispielsweise nach einem Reboot des Servers noch abrufen zu können. Um das zu erreichen, wird eine Datenbank benötigt, die sich um die Persistierung kümmert. Dabei wird zwischen zwei Arten von Datenbankkonzepten unterschieden: Relationale Datenbanken und NoSQL Datenbanken. Im folgenden werden beide Konzepte näher erläutert und ein Vergleich zwischen den beiden Arten erstellt.

#### 5.3.1.1 Relationale Datenbanken

#### 5.3.1.2 NoSQL Datenbanken

#### 5.3.1.3 Vergleich

### 5.3.2 Datenmodell

Um einen einheitlichen Zugriff auf Daten zu ermöglichen, muss ein Datenmodell entwickelt werden. Ein Datenmodell bezeichnet eine Struktur beziehungsweise den Aufbau der einzelnen Datensätze. Dabei ist zu beachten, dass in der Applikation Daten unterschiedlichster Art gespeichert werden. Das bedeutet, dass diese unterschiedlichen Typen logischerweise auch unterschiedliche Datenmodelle zugrunde haben müssen. Zur Organisation dieser unterschiedlichen Arten von Daten gibt es in MongoDB sogenannte Collections. Diese Collections beinhalten alle Datensätze eines Typs, beispielsweise alle Einträge von Benutzern. In unserer Applikation gibt es folgende Collections:

- **user**  
Zur Abspeicherung von Benutzeraccounts
- **notebook**  
Zur Abspeicherung von Schulheften; beinhaltet auch alle Daten innerhalb des Heftes (Texte, Bilder,...)
- **time\_table**  
Zur Abspeicherung eines Stundenplans; beinhaltet die einzelnen Stundenzeiten sowie eine Fachbezeichnung, einen Lehrer und einen Raum pro Tag und Stunde

### 5.3.2.1 User

Die Collection *user* beinhaltet alle relevanten Daten zur Abspeicherung von Benutzeraccounts. Folgendes Datenmodell liegt dem zugrunde:

- **\_id**  
Eine Object-ID zur eindeutigen Identifizierung eines Eintrags
- **email**  
Die E-Mail Adresse des jeweiligen Benutzers
- **first\_name**  
Der Vorname des jeweiligen Benutzers
- **last\_name**  
Der Nachname des jeweiligen Benutzers
- **password**  
Ein Hash eines selbst definierten Passwortes des jeweiligen Benutzers
- **is\_prouser**  
Beschreibt, ob der jeweilige Benutzer einen Pro-Account besitzt
- **is\_active**  
Beschreibt, ob der jeweilige Benutzer bereits seine E-Mail Adresse bestätigt hat
- **is\_superuser**  
Beschreibt, ob der jeweilige Benutzer ein Administrator der Applikation ist
- **last\_login**  
Das Datum, an dem der jeweilige Benutzer sich das letzte Mal erfolgreich an der Applikation angemeldet hat
- **date\_joined**  
Das Datum, an dem der jeweilige Benutzer sich an der Applikation registriert hat
- **validatetoken**  
Ein Hash, der als Token zur Bestätigung der E-Mail Adresse des jeweiligen Benutzers dient, sofern dieser die Bestätigung noch nicht durchgeführt hat
- **passwordreset**  
Ein Hash sowie ein Datum, welche zur Zurücksetzung des Passwortes des jeweiligen Benutzers dienen, sofern dieser angegeben hat, sein Passwort vergessen zu haben

Benutzer werden innerhalb der Applikation mithilfe ihrer E-Mail Adresse identifiziert. Das bedeutet, dass die Eigenschaft *email* bei jedem Eintrag in der Datenbank einzigartig sein muss, ebenso wie die ID des Eintrags.

### 5.3.2.2 Notebook

Die Collection *notebook* beinhaltet die Daten, die ein einzelnes Heft betreffen. Dazu wird folgende Struktur definiert:

- **\_id**  
Eine Object-ID zur eindeutigen Identifizierung eines Eintrags
- **name**  
Der Anzeigename des Heftes, der vom Benutzer festgelegt wurde
- **is\_public**  
Beschreibt, ob das Heft öffentlich (von allen Benutzern der Applikation) einsehbar ist
- **create\_date**  
Das Erstellungsdatum des Heftes
- **last\_change**  
Das Datum, an dem das Heft das letzte Mal bearbeitet wurde
- **email**  
Die E-Mail Adresse des Besitzers des Heftes
- **numpages**  
Die Anzahl an Seiten, die das Heft besitzt
- **current\_page**  
Die Seite, die aufgeschlagen wird, sobald der Benutzer das Heft das nächste Mal öffnet
- **content**  
Eine Liste, die alle Inhalte des Heftes (Texte, Bilder,...) beinhaltet
- **collaborator** Eine Liste an E-Mail Adressen von Benutzern der Applikation, die neben dem Besitzer des Heftes ebenfalls die Inhalte des Heftes bearbeiten dürfen

Der Name eines Heftes ist pro Benutzer einzigartig zu vergeben, um das Heft identifizieren zu können. Abgesehen von der Eigenschaft *\_id* kann ein Heft also auch mithilfe der beiden Eigenschaften *name* und *email* eindeutig identifiziert werden. Die Eigenschaft *content* besteht aus einer Liste. Diese Liste beinhaltet individuell viele JSON-Objekte, die jeweils ein Objekt innerhalb des Heftes darstellen (Text, Bild,...). Diese JSON-Objekte bestehen wiederum aus einer ID zur Identifizierung, der Art des Elementes (Textelement, Bildelement,...), der genauen Position innerhalb des Heftes (Seitenzahl und x-Koordinate, sowie y-Koordinate auf dieser Seite) und dem eigentlichen Inhalt, beispielsweise dem Text den der Benutzer eingegeben hat, sollte es sich um ein Textelement handeln.

### 5.3.2.3 Timetable

In der Collection *time\_table* werden alle Daten der Stundenpläne von Benutzern gespeichert. Das beinhaltet sowohl die einzelnen Fächer, Lehrer und Räume pro Stunde, als auch die per Benutzer definierten Zeiten für jede Stunde. Zur Persistierung wird folgendes Datenmodell verwendet:

- **\_id**  
Eine Object-ID zur eindeutigen Identifizierung eines Eintrags
- **email**  
Die E-Mail Adresse des Benutzers, dem der Stundenplan zugeordnet ist
- **times**  
Eine Liste, die die Anfangs- und Endzeiten jeder Stunde im Stundenplan enthält
- **fields**  
Eine Liste, die alle einzelnen Stunden im Stundenplan mit Fach, Lehrer und Raum enthält

Der Stundenplan wird einem Benutzer mithilfe der Eigenschaft *email* zugeordnet und damit auch eindeutig identifiziert, da jeder Benutzer nur einen Stundenplan besitzen kann.

Die Eigenschaften *times* und *fields* sind jeweils Listen, die mehrere JSON-Objekte enthalten. Ein JSON-Objekt in der Liste *times* enthält die jeweilige Stunde (1-xx) und die Anfangs- und Endzeit für diese Stunde. Ein JSON-Objekt in der Liste *fields* enthält eine ID zum Identifizieren der jeweiligen Stunde (Zusammengesetzt aus Reihenzahl und Spaltenzahl im Stundenplan), eine Bezeichnung des Faches, das in dieser Stunde unterrichtet wird, dem Namen des Lehrers, der das jeweilige Fach unterrichtet und dem Raum, in dem der Unterricht stattfindet, sowie ein Heftname eines Heftes, das dem jeweiligen Benutzer gehört. Mithilfe des Heftnamens kann ein Heft mit einer Stunde im Stundenplan verknüpft und zugeordnet werden.

### 5.3.3 Datenzugriff

Der Zugriff auf Daten in der Datenbank kann über zwei Arten erfolgen. Entweder es wird direkt über die Konsole von MongoDB zugegriffen, oder der Datenzugriff erfolgt über die Applikation.

Der Zugriff auf Daten bezeichnet dabei unterschiedliche Operationen. Es können neue Daten erstellt, vorhandene Daten ausgelesen und bestehende Daten bearbeitet oder gelöscht werden.

**5.3.3.1 Direkter Zugriff auf die Datenbank**

**5.3.3.2 Zugriff aus der Applikation**



## 5.4 Parallel Working System

Bereits bestehende PWS Systeme

Umsetzungsversuch auf unser Projekt angepasst

Vorgehensweise

Verbesserungsvorschläge

Ausblick

## 5.5 Optical Character Recognition

## 6 Auswertung und Benchmarks

## 7 Ausblick

Bezahlungssystem

PDF Download

LDAP Anbindung

Pro Accounts

Stundenplan Web Untis Import

Zeichentool

Video einbindung

Filehoster anbindung

Shortcuts für schnellere Verwendung

...

## 8 Zusammenfassung

## 9 Anhang

### 9.1 Glossar

<b>API</b> Application Programming Interface.....	12
<b>DNS</b> Domain Name Service .....	10
<b>HTTP</b> Hypertext Transfer Protocol.....	10
<b>HTTPS</b> Hypertext Transfer Protocol Secure .....	10
<b>IDE</b> Integrated Development Environment .....	13
<b>KVM</b> Kernel-based Virtual Machine .....	9
<b>Latenz</b> Verzögerung zwischen Anfrage und Antwort(Ping) .....	8
<b>SSH</b> Secure Shell.....	9
<b>SSL</b> Secure Sockets Layert .....	12
<b>Stakeholder</b> Eine Person welche Interesse am Ergebnis des Projektes hat, jedoch nicht an der Entwicklung des Projektes an sich beteiligt ist.....	8
<b>TGM</b> Technologisches Gewerbe Museum .....	9
<b>WSGI</b> Web Server Gateway Interface .....	13

## 9.2 Abbildungen

1	Loadbalancing [Cep14]	11
2	Deployment Struktur (selfmade)	13
3	Functional Testing Level [Bur03]	15
4	Authentisierung bei DSN	18
5	Auslösen des Captchas	20
6	Klassische Anmeldung oder mittels OAuth	22
7	Navigationsleiste als User	23
8	Kontoeinstellungen	24
9	Usersuche	24

## 9.3 Listings

1	Firewall Rules des Projektserver	10
2	Installation iptables-persistent	11
3	Selenium Element Selektoren	16
4	Selenium Test	17
5	Einbindung der JS-Library Recaptcha	19

## 9.4 Quellen

- [AST08] Maarten van Steen Andrew S. Tanenbaum. *Verteilte Systeme*. PEARSON, 2 edition, 2008.
- [Bur03] Ilene Burnstein. Software testing level chart. [http://images.books24x7.com/bookimages/id\\_16280/fig6-1.jpg](http://images.books24x7.com/bookimages/id_16280/fig6-1.jpg), 2003.
- [Cep14] Lukasz Cepowski. Loadbalancing with nginx. <http://cepa.io/image/6>, 2014.
- [DCH15] Dr. Dr. Jörg Berwanger Dr. Cordula Heldt. Allgemeine geschäftsbedingungen (agb). <http://wirtschaftslexikon.gabler.de/Archiv/5433/allgemeine-geschaeftsbedingungen-agb-v9.html>, 2015.
- [DI15] Klaus Lipinski Dipl.-Ing. OAuth(open authentication). <http://www.itwissen.info/definition/lexikon/OAuth.html>, 2015.
- [Fou15] Django Software Foundation. Writing custom django-admin commands. <https://docs.djangoproject.com/en/1.9/howto/custom-management-commands/>, 2015.
- [Git15] Github. Mongoengine/mongoengine auth.py. <https://github.com/MongoEngine/mongoengine/blob/0.9/mongoengine/django/auth.py>, 2015.
- [Kha15] Ashish Khatkar. Issue with mongo in django. <http://comments.gmane.org/gmane.comp.python.django.user/171657>, 2015.

- [Kre15a] Thomas Krenn?? Brute-force-angriffe?? [http://www.password-depot.de/know-how/brute\\_force\\_angriffe.htm](http://www.password-depot.de/know-how/brute_force_angriffe.htm), 2015.
- [Kre15b] Thomas Krenn. Iptables firewall regeln dauerhaft speichern. [https://www.thomas-krenn.com/de/wiki/Iptables\\_Firewall\\_Regeln\\_dauerhaft\\_speichern](https://www.thomas-krenn.com/de/wiki/Iptables_Firewall_Regeln_dauerhaft_speichern), 2015.
- [Pea15] LaTonya Pearson. The four levels of software testing. <http://www.seguetech.com/blog/2013/07/31/four-levels-software-testing>, 2015.
- [Sha15] Rahil Shaiksh. Google recaptcha with angularjs. <http://code.ciphertrick.com/2015/05/19/google-recaptcha-with-angularjs/>, 2015.
- [tut15] tutorialspoint. Software testing - levels. [http://www.tutorialspoint.com/software\\_testing/software\\_testing\\_levels.htm](http://www.tutorialspoint.com/software_testing/software_testing_levels.htm), 2015.
- [ubu15] ubuntuusers. Cron. <https://wiki.ubuntuusers.de/Cron/>, 2015.