



TGM - HTBLuVA Wien XX
IT Abteilung

Diplomarbeit
DigitalSchoolNotes

Inhalt

1	Einleitung	1
2	Problembeschreibung	1
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
3	Stand der Technik	3
3.1	Frameworks	3
3.2	Technologien	4
4	Design	5
4.1	Software-Architektur	5
4.2	Graphische Oberfläche	6
4.3	Javascript Optimierung	7
5	Implementierung	8
5.1	Infrastruktur und Testing	8
5.1.1	Infrastruktur	8
5.1.2	Testing	11
5.2	User und Rollenmanagement	12
5.2.1	Usermanagment	12
5.2.2	Authentisierung	12
5.2.3	Datenmodell	14
5.2.4	Email	17
5.2.5	Anmelden	18
5.2.6	Usersicht	19
5.2.7	Adminsicht	20
5.3	Datenmanagement	25
5.4	Parallel Working System	26
5.5	Optical Character Recognition	27
6	Auswertung und Benchmarks	28
7	Ausblick	29
8	Zusammenfassung	30

9	Anhang	i
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
```

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
```

5.1.1.7 Bruteforce Prevention

5.1.1.8 Webserver

5.1.1.9 SSL

5.1.1.10 Produktivbetrieb

5.1.1.11 Testbetrieb

5.1.1.12 Verfügbarkeit

5.1.2 Testing

5.2 User und Rollenmanagement

5.2.1 Usermanagement

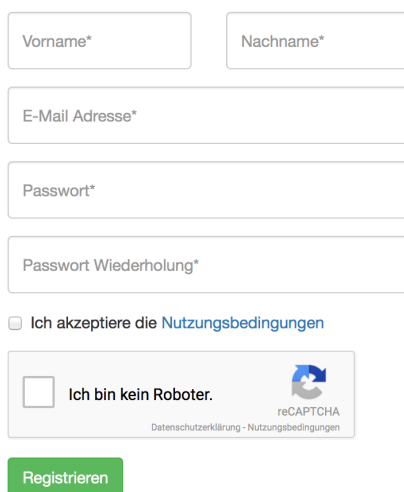
Unter dem Begriff Usermanagement versteht sich, dass verwalten und kontrollieren von Benutzerkonten. Es soll dazu dienen jeden registrierten User eindeutig zu identifizieren und zu kontrollieren, ob die monatlichen Raten für einen Pro-Account überwiesen wurden. Außerdem soll die bereits in Anspruch genommene 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 DigitalSchoolNotes. It consists of several input fields and checkboxes. At the top, there are two side-by-side text boxes labeled 'Vorname*' and 'Nachname*'. Below these is a single text box for 'E-Mail Adresse*'. This is followed by two more text boxes for 'Passwort*' and 'Passwort Wiederholung*'. Below the password fields is a checkbox labeled 'Ich akzeptiere die Nutzungsbedingungen'. At the bottom, there is a reCAPTCHA widget with the text 'Ich bin kein Roboter.' and a small robot icon. Below the reCAPTCHA is a link for 'Datenschutzerklärung - Nutzungsbedingungen'. Finally, at the very bottom, there is a green button labeled 'Registrieren'.

Um auszuschließen, dass sich eine Software bzw. ein Roboter einen Account auf DSN erzeugt, wird ein Captcha verwendet. Ein Captcha dient zur Sicherheit und soll überprüfen wer die Eingabe getätigt hat. Das Captcha muss serverseitig validiert werden. Mit den Daten des Formulars 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 ist, also ob der User ein menschliches Lebewesen ist. 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."
```

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>
```

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 nun angezeigt. Um zu überprüfen ob es ausgefüllt wurde folgendes unternommen:

```
if(vcRecaptchaService.getResponse() === ""){
    $scope.captchaerror = true;
    $scope.captcha_error = "Bitte loese das Captcha.";
}
```

Wenn das Captcha gelöst wurde, kann man damit das Form genau ein Mal absenden. Falls es nochmal abgesendet werden muss, muss das Captcha zurückgesetzt werden. Dies kann wie folgt umgesetzt werden:

```
vcRecaptchaService.reload();
```



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.

Im File settings.py muss zu *INSTALLED_APPS* *'mongoengine.django.mongo_auth'* hinzugefügt werden. Außerdem muss im selben File folgender Code eingefügt werden:

```
AUTHENTICATION_BACKENDS = (  
    'mongoengine.django.auth.MongoEngineBackend',  
)  
  
AUTH_USER_MODEL=('mongo_auth.MongoUser')  
MONGOENGINE_USER_DOCUMENT = 'dsn.models.User'
```

Nun muss das User-Model, das eben 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.

```
from mongoengine.django.auth import UserManager, Permission, \
    make_password, check_password, SiteProfileNotAvailable, \
    _user_get_all_permissions, _user_has_module_perms, _user_has_perm
from mongoengine.django import auth
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))

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['last_name', 'first_name']

    meta = {
        'allow_inheritance': True,
        'indexes': [
            {'fields': ['email'], 'unique': True, 'sparse': True}
        ]
    }

[...]

    @classmethod
    def create_user(cls, email, password, first_name, last_name):
        now = datetime.datetime.now()

[...]
        user = cls(id=ObjectId(), email=email, date_joined=now, \
                    first_name=first_name, last_name=last_name)
        user.set_password(password)
        user.save()
        return user

[...]
```

Außerdem muss in *models.py* noch ein UserManager erstellt werden:

```
class AuthUserManager(UserManager):
    def create_user(self, email, password, first_name, last_name):
        if email and password:
            try:
                User.objects.get(email=email)
                return None
            except DoesNotExist:
                try:
                    email_name, domain_part = email.strip().split('@', 1)
                except ValueError:
                    pass
                else:
                    email = '@'.join([email_name, domain_part.lower()])

                user = User(username=email, email=email, \
                            first_name=first_name, \
                            last_name=last_name)
                user.set_password(password)
                user.save()
                return user
            else:
                return None

    def create_superuser(self, email, password, first_name, last_name):
        if email and password:
            try:
                User.objects.get(email=email)
                return None
            except DoesNotExist:
                try:
                    email_name, domain_part = email.strip().split('@', 1)
                except ValueError:
                    pass
                else:
                    email = '@'.join([email_name, domain_part.lower()])

                user = User(email=email, is_superuser=True, \
                            first_name=first_name, \
                            last_name=last_name)
                user.set_password(password)
                user.save()
                return user
            else:
                return None
```

Da MongoDB den PrimaryKey als ObjectId verlangt, muss 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` 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`

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 man Email-Adresse des Benutzer 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 nächsten 24 Stunden verwendet wird, dann ist der Zeitstempel abgelaufen und es muss ein neuer angefordert werden. Hingegen die Email Adresse dient dem Nachweis, welcher Account aktiviert wird. Im File `authentication/registration.py` in Zeile 23:

```
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. Als erstes muss im `settings.py` 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 = 'password'
EMAIL_PORT = 25
EMAIL_USE_TLS = False
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```

5.2.5 Anmelden

Wenn der neue Benutzer den Registrierprozess erfolgreich absolviert hat, steht es 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 Webservice 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


E-Mail Adresse

Passwort

Anmelden [Passwort vergessen?](#)

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

 Sign in with Facebook

 Sign in with Google

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



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

Kontoeinstellungen

Vorname

Nachname

E-Mail Adresse

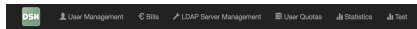
Aktuelles Passwort*

Passwort ändern

Neues Passwort

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.

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 angegebenen 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@y.vom	y	y	Inaktiv	Account löschen	Mail senden
xx@x.com	x	x	Inaktiv	Account löschen	Mail senden
vorname928@nachname928.test	Vorname928	Nachname928	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

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

9.2 Abbildungen

9.3 Listings

9.4 Quellen

- [AST08] Maarten van Steen Andrew S. Tanenbaum. *Verteilte Systeme*. PEARSON, 2 edition, 2008.
- [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, 2015.
- [Kre15b] Thomas Krenn. Iptables firewall regeln dauerhaft speichern. https://www.thomas-krenn.com/de/wiki/Iptables_Firewall_Regeln_dauerhaft_speichern, 2015.
- [Sha15] Rahil Shaiksh. Google recaptcha with angularjs. <http://code.ciphertrick.com/2015/05/19/google-recaptcha-with-angularjs/>, 2015.
- [ubu15] ubuntuusers. Cron. <https://wiki.ubuntuusers.de/Cron/>, 2015.