Inhaltsverzeichnis

1 Einführung

Aktuell ist in Blattwerkzeug keine Benutzer-Authentisierung, -Authentifizierung und -Autorisierung implementiert. Dies hat zur Folge, dass zum jetzigen Zeitpunkt jeder Blattwerkzeug-Nutzer dazu autorisiert ist, jegliche, vom Client erlaubten, Änderungen vorzunehmen. Dies resultiert aus dem bisher einzigen Nutzer in der Datenbank. Das Adminpanel ist beispielsweise für jeden Blattwerkzeug-Nutzer, über die Seiten-Navigation, frei zugänglich. Im Rahmen dieser Thesis soll genau diesesProblem gelöst werden. Nach Behandlung der Thesis soll es möglich sein, sich mit einer standardisierten Registrierung bei Blattwerkzeug anzumelden. AuSSerdem soll es ebenfalls möglich sein, sich über externe Anbieter anzumelden. Zusätzlich soll je nach Benutzerrolle und Benutzergruppe des angemeldeten Nutzers unterschiedlicher Inhalt dargestellt werden. Darüber hinaus soll ein angemeldeter Benutzer die Möglichkeit haben sein bereits erstelltes Konto mit weiteren E-Mails oder externen Konten zu verknüpfen. In den Einstellungen soll es dem Nutzer auSSerdem ermöglicht werden seine verknüpften Konten zu verwalten.

Der Server erlaubt, der Client bietet an

du löst noch mehr

2 Technologien

Im Verlauf dieser Sektion werden die Technologien und deren Verwendungszweck kurz erläutert.

2.1 Blattwerkzeug

Blattwerkzeug ist ein quelloffenes Projekt, dass Informatik-Interessierten das Programmieren von Hypertext Markup Language (HTML) Grundgerüsten und SQL Statements per „drag and drop“ näher bringen kann. Dabei versteckt Blattwerkzeug die Syntax nicht vor dem Nutzer, sondern gibt ihm die Möglichkeit diesen gleich mit einzusehen. Dennoch ist es dem Nutzer einfach gemacht, mit visuellen Elementen teile der Informatik kennen zu lernen.

Dabei hat es sich Blattwerkzeug vor allem als Aufgabe gemacht an Schulen aufzutreten. Mit Blattwerkzeug wird Lehrern ein Werkzeug in die Hand gelegt, mit dem der Informatik Unterricht einfacher und informativer gestaltet werden kann. Somit wird der veraltete

und doch sehr Office-lastige Informatik Unterricht komplett erneuert und interessanter gestaltet werden.

Zuviel: Blatt-Werkzeug ist ein Zusatz, keine Ersetzung

2.2 Passwort Hashing

Sobald eine Software mit Nutzerdaten geführt wird, ergibt sich das Problem des Speicherns der Passwörter der jeweiligen Nutzer. Denn sollten die Daten der Nutzer im Klartext in der Datenbank gespeichert werden und ein Angreifer erlangt Zugriff auf die Datenbank, so ist es für ihn ein leichtes weitere Konten der Nutzer zu infiltrieren. Der Grund dafür sind die anwendungsübergreifenden, vom Nutzer größtenteils identischen, Passwörter.

Nicht nur Super-GAU für eigene Seite, sondern auch für Nutzer auf anderen Seiten. Konkretes Beispiel ergänzen (Liesschen Müller ist mit ihrer EMail „lieeschen@müller.de“ und dem Passwort „Milch“ bei „Milchkanne.de“ registriert, ...

An diesem Punkt kommt das Hashen von Passwörtern zum Einsatz. Passwort Hashing soll dem Nutzer Sicherheit gewährleisten und es einem Angreifer nicht möglich machen mit erlangten Daten weitere Konten der Nutzer zu infiltrieren. Dabei wird aus einem Passwort ein Hash generiert. Dieser Hash macht es einem unmöglich, das Passwort wiederherzustellen. Jedoch ergibt sich bei gleicher Eingabe, der gleiche Hash. Um ein gehashtes Passwort zu erhalten, muss ein Hashing Algorithmus auf das jeweilige Klartext Passwort angewendet werden.

Mittlerweile gibt es eine vielzahl an Hashfunktionen, von denen manche als nicht mehr sicher gelten. Der Grund dafür sind Rainbowtables in denen Hashes mit dazugehörigem Klartext Passwort (Abbildung 1) stehen. Das Nutzen einer nicht sicheren Hashfunktion stellt ein Sicherheitsrisiko dar, da die Möglichkeit besteht, das gehashte Passwort mit einer Rainbowtable (Abbildung 4) abzugleichen und dabei das jeweilige Klartext Passwort zu erhalten. Aus diesem Grund werden MD5 (Abbildung 2) und SHA zwei der bekanntesten Hashfunktionen, seit geraumer Zeit nicht mehr zum Passwort hashen verwendet werden. Ein Beispiel für eine derzeitig nutzbare Hashfunktion ist bcrypt. Bcrypt bietet den Vorteil des Integrierten Saltings. (Abbildung 3)

Salts (Abbildung 5) sind zufällig generierte Zeichenketten, die dem Klartext-Passwort hinzugefügt werden. Der zusammengesetzte String wird mittels Hashfunktion verschlüsselt.

id	email	password
1	tom@blattwerkzeug.de	a7dAe2dvQ2
2	marcus@blattwerkzeug.de	?3s2Dq2q1!
3	chantal@web.de	12345678

Abbildung 1: Tabelle mit ungehashten Passwörtern

id	email	password
1	tom@blattwerkzeug.de	133e7ed9176f9ddc692d00f1bc205bb9
2	marcus@blattwerkzeug.de	3a93dd58b4761acaecc9fccf795aa450
3	chantal@web.de	25d55ad283aa400af464c76d713c07ad

Abbildung 2: Tabelle mit MD5 gehashten Passwörtern

id	email	password
1	tom@blattwerkzeug.de	$2y$12$9T9p6L8zfm4H71Aezi91jO9zwnZ/aCh9koU9jZxZ1IfInjelUorzi
2	marcus@blattwerkzeug.de	$2y$12$rz2WMylALzCRD2n0o0PFo.AdAyFRW0QYlOCGZMFYDqj2dHORK.S7K
3	chantal@web.de	$2y$12$weQa6P6JDy.AGvjlQqtS..QEybFSuBoFVhWfXkxRtH5ga7KxpGer.

Abbildung 3: Tabelle mit Salting Hashes

hashed	unhashed
133e7ed9176f9ddc692d00f1bc205bb9	a7dAe2dvQ2
3a93dd58b4761acaecc9fccf795aa450	?3s2Dq2q1!
25d55ad283aa400af464c76d713c07ad	123456678

Abbildung 4: Beispiel einer Rainbowtable.

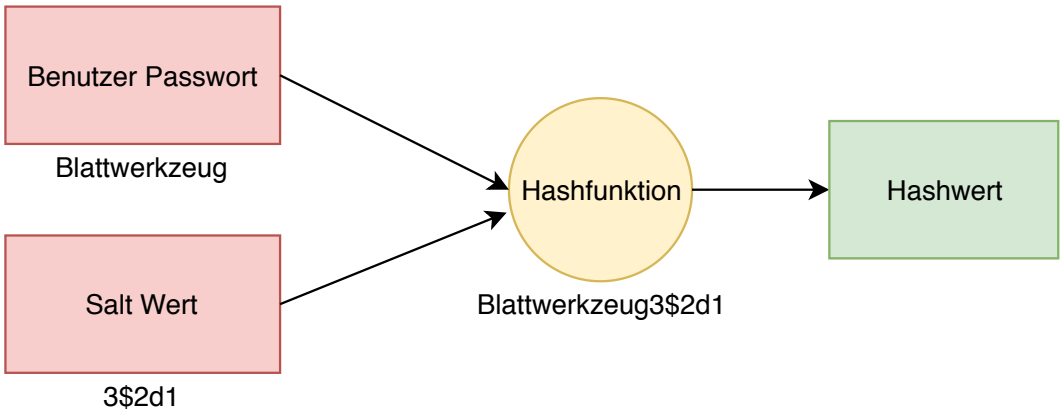


Abbildung 5: Hashfunktion auf Klartext und Salt angewandt.

2.3 Sessions

Das Hypertext Transfer Protocol (HTTP) ist ein zustandsloses Protokoll, dass sich keine Informationen der jeweiligen Aufrufe zwischenspeichert. Dies ist aber unpraktisch, wenn Daten eines Benutzer kurzeitig gespeichert werden sollen. Ein Verwendungszweck wäre beispielsweise der Warenkorb, da dieser nur temporär vorhanden sein soll. Genau dieses Problem kann mit der Session gelöst werden.

Zustandslos prak-tisch

Warum lösen Coo-kies das Pro-blem nicht?

Die Session ist eine serverseitige Daten-Speichermöglichkeit. Dabei wird bei der Anfrage von einem Client an den Server ohne Session-ID eine Session und Session-ID erstellt. Diese Session-ID wird bei der Antwort des Servers mit an den Client ausgeliefert. Ab diesem Punkt wird bei jeder Anfrage vom Client an den Server die Session-ID mit gesendet. Dies kann über einen Cookie oder über die Uniform Resource Identifier (URI) erfolgen. Aufgrund dessen kann der Server dem Client Daten aus der jeweiligen Session zur Verfügung stellen.

IMAGE

2.4 JSON Web Token

„JWT sind auf JavaScript Object Notation (JSON) basierende Request for Comments (RFC) 7519 genormte Access-Token.“ RFC ist eine Sammlung aus Dokumenten, in denen das Verhalten der Technologien des Internets beschrieben ist. Einige davon gehören zum Standard und werden somit in den meisten Fällen vorausgesetzt. In speziellen Fällen möchte beispielsweise ein Unternehmen eigene Protokolle verwenden, die nicht zum Standard gehören.

Diese Tokens werden zur eindeutigen Identifizierung von Nutzern verwendet und können die Session ersetzen. Dabei ist es bei einem JWT nicht vonnöten die Daten auf dem Server zu speichern. Dies hat zur Folge, dass die Pflege des Speichers an diesem Punkt entfällt. Jedoch haben JWTs einen großen Nachteil, denn sobald der Server einen JWT ausgestellt hat, ist dieser bis zum Ablauf des Tokens gültig. Das heißt, sollte ein Server die Berechtigung eines Nutzers nach Ausstellung eines JWT ändern, ist diese Änderung erst bei erneutem Erstellen eines JWT gültig.

Ein JWT besteht aus Header, Payload und Signatur. Dabei ist der Header und die Payload jeweils ein JSON Objekt.

2.4.1 Header

typ Der typ Claim beschreibt den Internet Media Type (MIME) des JWT, dieser wiederum teilt dem Client oder Server mit, um welche Art von Medium an Daten es sich handelt. Der Standardwert dieses Claims beläuft sich auf „JWT“, übersetzt „application/jwt“.

alg Der alg Claim beschreibt die Verschlüsselungsmethode. Ein Beispiel ist Keyed-Hash Message Authentication Code (HMAC) mit Secure Hash Algorithm 256 Bit (SHA256), HS256 abgekürzt.

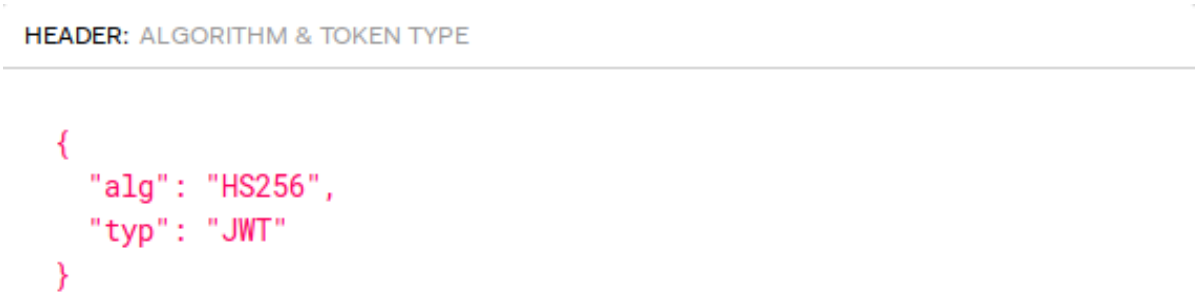


Abbildung 6: Beispiel eines JWT Headers

2.4.2 Payload

Die Payload beinhalteten Schlüssel-Wert Paare werden Claims genannt. Dabei handelt es sich um ein JSON Objekt, bei dem bestimmte Schlüssel des Objektes bereits reserviert sind. Diese nennen sich registrierte Claims. Außerdem gibt es öffentliche und private Claims. Hierbei wird zwischen öffentlichen und privaten differenziert.

Beispiel registrierter Claims

iss Der iss Claim steht für den Austeller des Tokens, beispielsweise eine Domain.

exp Der exp Claim kennzeichnet den JWT mit einem Ablaufdatum.

Öffentliche Claims

Öffentliche Claims sind zusätzlich zum Standard nutzbar und ihre Namen sollten semantisch dem dazugehörigen Wert entsprechen. Außerdem sollten die Namen der Claims Netzwerkübergreifend verständlich sein (Listing 1).

Private Claims

Private Claims werden nur innerhalb eines Netzwerkes verwendet. Aus diesem Grund gibt es keine implizite Beschränkung in der Namensgebung (Listing 2).

```
{
    "name": "Tom Hilge",
    "email": "Blattwerkzeug@web.de"
}
```

Listing 1: Beispiel eines Öffentlichen Claims

```
{
    "blattwerkzeug_role": "test"
}
```

Listing 2: Beispiel eines Privaten Claims

2.4.3 Signatur

Um die Signatur zu erhalten muss, die Payload und der Header Base64 kodiert werden. Außerdem müssen diese beiden kodierten Zeichenfolgen mit einem Punkt als Trennzeichen verknüpft werden. Darauffolgend wird eine Hashfunktion auf das jeweilige Ergebnis mit zusätzlich sicherer Zeichenfolge als Parameter angewandt. Da diese sichere Zeichenfolge, auch Private Key genannt, nur auf dem Server hinterlegt ist, ist es dem Client zwar möglich den JWT zu verändern, ihn jedoch mit korrekter Signatur zu versehen nicht.

2.4.4 Zusammengesetzes Token

Schlussendlich ergibt sich der JWT aus kodiertem Header, kodierten Payload und der Signatur. Dabei steht der Header am Anfang (Abbildung 7, rot gekennzeichnet). Darauffolgend mit einem Punkt getrennt die Payload und zum Schluss die Signatur, ebenfalls mit einem Punkt getrennt.

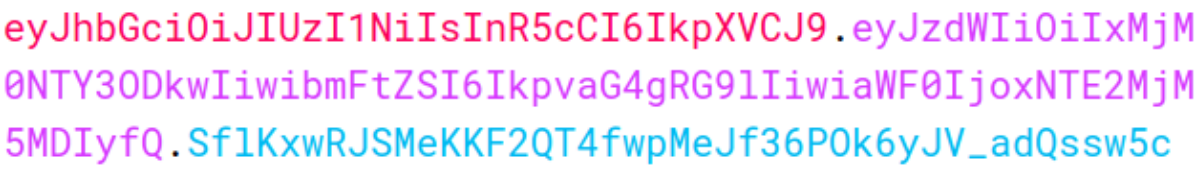


Abbildung 7: Beispiel eines kodierten JWT

2.5 Ruby on Rails

Ruby on Rails ist ein quelloffenes Webframework für die Programmiersprache Ruby. Das Webframework nutzt das Model View Controller (MVC) Muster und stellt bereits ein sehr umfangreiches Command Line Interface (CLI) zur Verfügung. Mittels des generate Werkzeugs kann beispielsweise Model, View und Controller erstellt werden. Jeder dieser Komponenten wird automatisch in die erstellte Rails Anwendung eingebunden. Au/sserdem stellt Rails eine umfangreiche Test-Architektur und einen Service zum Versenden von Mails zur Verfügung. Dabei kann der Inhalt der E-Mail im Textformat oder als HTML versendet werden. Einer der wesentlichen Vorteile von Ruby on Rails ist jedoch die Datenbankanbindung. Hierbei bietet Rails einen nachhaltigen und rücksichtsvollen Umgang mit der Datenbank, zum Beispiel die Migrationen. Migrationen erlauben, die Datenbank, ohne explizite SQL-Statements, zu verändern. Zusätzlich erleichtern Migrationen die Implementierung einer Datenbankstruktur auf einem anderen System.

2.5.1 Routen

Die Routen in Rails verweisen auf einen Controller und auf eine Funktion innerhalb des Controllers. Dabei wird die Route meistens mit der Anfragemethode eingeleitet, beispielsweise „get“. Routen können in sogenannte „scopes“ (Listing 4) unterteilt werden. Somit ist es nicht vonnöten bei einer verschachtelten URI redundant zu werden (Listung 3).

```
get 'news/admin', controller: 'news', action: :index_admin
get 'news/admin/:id', controller: 'news', action: :show_admin

get 'news', controller: 'news', action: :index
get 'news/:id', controller: 'news', action: :index
post 'news', controller: 'news', action: :create
put 'news/:id', controller: 'news', action: :update
delete 'news/:id', controller: 'news', action: :delete
```

Listing 3: Beispiel einiger redundanter Routen

```
scope 'news' do
    scope 'admin' do
        root via: [:get], controller: 'news', action: :index_admin
        get ':id', controller: 'news', action: :show_admin
    end

    root via: [:get], controller: 'news', action: :index
    root via: [:post], controller: 'news', action: :create
    get ':id', controller: 'news', action: :show
    put ':id', controller: 'news', action: :update
    delete ':id', controller: 'news', action: :destroy
end
```

Listing 4: Beispiel einiger Routen mit scope

2.5.2 Controller

Der Controller dient hierbei zur Kapselung von bestimmten Prozessen. Jede Route verweist in irgendeiner Weise auf eine Controller Funktion. In der jeweiligen Controller Funktion wird dann meistens mit einem Model interagiert. Es wird beispielsweise eine Benutzerberechtigung abgefragt und individuell auf die Berechtigung reagiert. Um auf die jeweilige Berechtigung zu reagieren, gibt es mehrere Möglichkeiten. Eine der Möglichkeiten wäre, direkt ein View Template auf dem Server zu rendern und an den Client auszuliefern. Eine andere Möglichkeit wäre ein JSON Objekt zurück zu geben und darauf mit dem Client zu agieren.

2.5.3 Model

Das Model in Rails stellt jeweils eine Datenbanktabelle dar. Die Attribute des Models sind die entsprechenden Spalten der Datenbanktabelle. Jeweilige Datenbankeinträge, die

über das Model erstellt werden, können mittels Validatoren auf ihre Gültigkeit geprüft werden. Diese Validatoren werden innerhalb des Models festgelegt und auf ein Attribut des Models zugewiesen. Rails bietet dabei bereits verfügbare Validatoren, zum Beispiel „presence: true“. Dieser Validator sorgt für das Vorhandensein eines Wertes ungleich *nil*. Jedes Model kann zusätzliche Funktionen beinhalten, die direkt auf den jeweiligen Datenbankeintrag angewandt werden können. Ebenso bietet Rails die Möglichkeit die Beziehungen zwischen Datenbanktabellen direkt in den Modellen festzulegen.

2.5.4 View

Die View stellt in Rails die Möglichkeit HTML Template auf dem Server zu rendern. Dabei kann bei dem Rendern das HTML Template dynamisch verändert werden. Da diese Komponente während dieser Thesis keine Rolle gespielt hat, wird diese nicht weiter erläutert.

2.5.5 Zusammenfassung

Letzendlich wird über die Route auf den jeweiligen Controller zugegriffen. Dieser fragt in den meisten Fällen nach einem bestimmten Eintrag eines Models. Darauffolgend wird mit dem Ergebnis der Anfrage interagiert. Es werden Veränderungen oder Abfragen bestimmter Daten getätigt. Danach wird ein Ergebnis dem Client ausgeliefert.

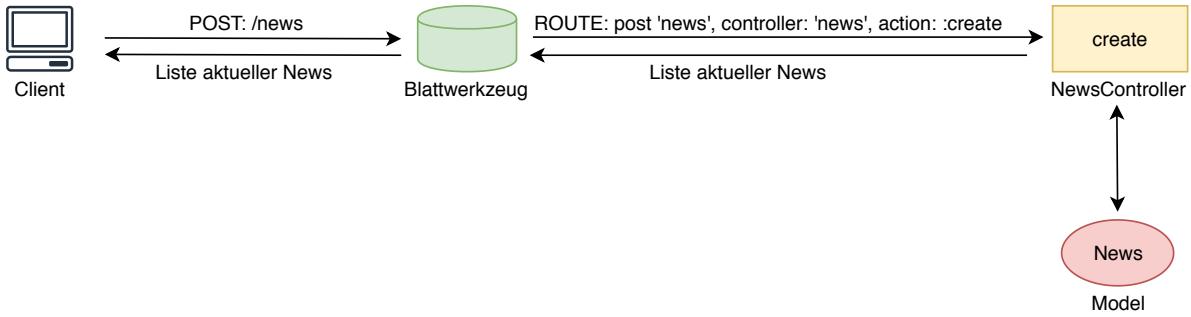


Abbildung 8: Ablauf zwischen Client, Server, Route, Controller und Model

2.6 Angular

Angular ist ein TypeScript basiertes Front-End Webframework, dass in vielen Fällen für Single Page Application (SPA) verwendet wird. SPAs laden ihren Inhalt lediglich in ein einziges HTML Dokument. Der Inhalt dieses HTML Dokumentes wird dynamisch, von beispielsweise einem Framework wie Angular, verändert. Der wesentliche Vorteil von Angular sind die klaren Entwurfsmuster. Jede Komponente in Angular hat im wesentlichen

die gleiche Struktur. Dies hat zur Folge, dass Angular eine sehr gute Codekonsistenz bietet.

2.6.1 Component

Komponenten in Angular bieten die Möglichkeit HTML, Cascading Style Sheets (CSS) und TypeScript zu kapseln. Das bedeutet, dass jede Komponente unabhängig von einer anderen Komponente arbeiten kann.

2.6.2 Services

Zur Kommunikation mit einem Server und/oder zum Datenaustausch zwischen unterschiedlichen Komponenten wird meistens ein Service verwendet. Bei einem Datenaustausch zwischen Eltern- und Kind-Komponente ist es jedoch einfacher dies mittels der Kind-Komponente durchzuführen. Services werden bei dem Laden der Module instanziiert und dem Konstruktor der Komponente als instanziiertes Objekt übergeben.

2.6.3 Module

Zusätzlich bietet Angular auch die Möglichkeit eigene Module zu erstellen in denen dann als Beispiel Services und Komponenten zusätzlich abgekapselt werden können. Ein Vorteil von Angular gegenüber anderen JavaScript Frameworks sind die bereits von Angular mitgelieferten Module, beispielsweise das Routing- oder das HTTP-Modul. Das Routing-Modul wird für jegliche Navigation auf der Anwendung genutzt. Das HTTP-Modul hingegen bietet die Möglichkeit mittels jeglicher Anfragemethoden mit dem Server zu kommunizieren.

2.7 oAuth2

oAuth2 ist ein offenes RFC 6749 Protokoll, welches verwendet wird, um eine Authentifizierung einer Anwendung mittels Drittanbieter zu ermöglichen. Hierbei wird der Nutzer zuerst auf die jeweilige Seite des Drittanbieters weitergeleitet. Dort muss der Nutzer sich authentifizieren und den Zugriff auf die Daten seines Kontos bestätigen. Nachdem der Zugriff auf die Daten bestätigt wurde, erhält die jeweilige Anwendung von dem Drittanbieter einen Autorisierungstoken. Dieser Autorisierungstoken wird darauffolgend von der Anwendung genutzt, um einen Zugriffstoken von dem Drittanbieter zu erhalten. Dieser ermöglicht am Ende den Zugriff auf die spezifischen Nutzerdaten des Drittanbieters. (Abbildung 9)

In Blattwerkzeug wird genau dieser umfangreiche Vorgang von Omniauth übernommen. Aus diesem Grund wird oAuth2 in dieser Thesis nicht weiter erläutert.

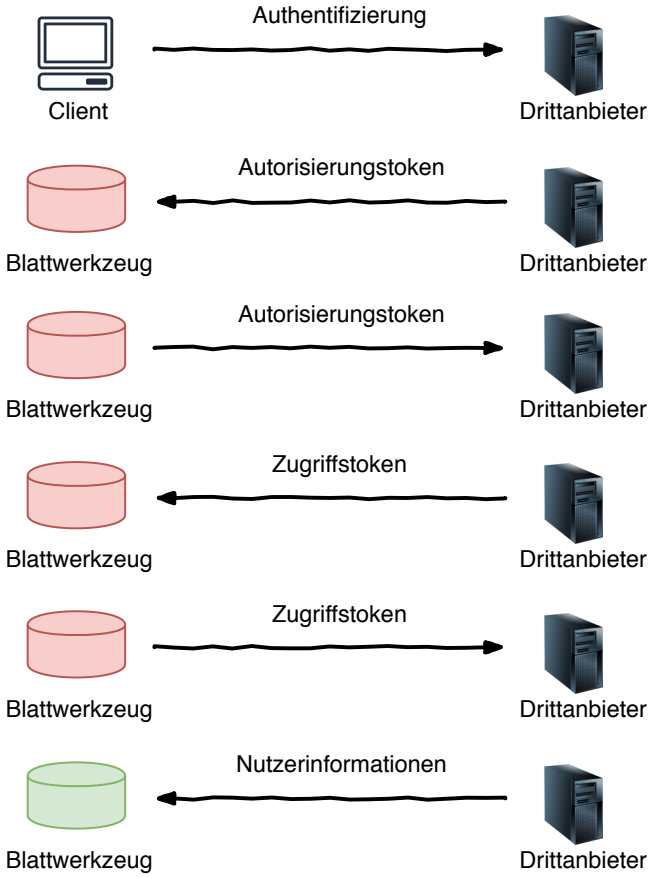


Abbildung 9: oAuth2 verfahren

3 Anforderungsanalyse

Das Ziel dieser Thesis ist, ein standardisiertes Registrierungsverfahren zu erstellen, zuzüglich einer Authentifikation über oAuth2 oder ein Passwort. Hinzu kommt eine Möglichkeit, den Nutzer über den Server zu autorisieren und ihm nach spezifischen Benutzerrollen unterschiedlichen Inhalt zu präsentieren. Dabei soll es dem Nutzer nicht gestattet sein, durch Manipulation seines Clients mit verfälschten Daten, Zugriff auf für ihn nicht zugreiffbare Daten zu erhalten. Um das Ziel dieser Thesis zu erreichen, muss man sich vorerst mit Ruby und dem Webframework Rails auseinandersetzen. Zudem ist es vonnöten, sich intensiver mit Angular zu beschäftigen, da dieses bisher nur oberflächlich behandelt wurde.

3.1 Derzeitiges Projekt

Zum derzeitigen Zeitpunkt ist Blattwerkzeug eine Lernplattform, in der jeder Nutzer jedes Projekt bearbeiten kann. Dafür wurde serverseitig ein Benutzername und Passwort, in beiden Fällen „user“, hinterlegt. Zusätzlich ist es möglich, das Adminpanel ohne weitere Autorisierungsabfrage zu betätigen.

3.1.1 Client

Der clientseitige Teil von Blattwerkzeug basiert auf Angular, Angular-Material und Bootstrap, hierbei sind Angular-Material und Bootstrap Gestaltungsframeworks. Bootstrap, welches hauptsächlich auf CSS und HTML basiert und Angular-Material, dass explizit Module für Angular bereitstellt.

3.1.2 Server

Der serverseitige Teil baut auf Ruby on Rails und bietet ein Application Programming Interface (API) zur Kommunikation mit dem Server. Die Daten werden mittels unterschiedlicher Anfragemethoden abgefragt, übermittelt und vom Server verarbeitet.

3.2 Anforderungen

Im Verlauf dieser Sektion werden die Anforderungen, die diese Thesis erfüllen soll, detailliert erläutert.

3.2.1 Unterschiedliche Anmeldemöglichkeiten

Ein Benutzer sollte die Möglichkeit haben sich mit mehreren Konten zu verknüpfen. Das bedeutet, einem Benutzer würde die Möglichkeit gegeben sein, sich mit Passwort und zum Beispiel Google anzumelden.

3.2.2 Anmeldung mittels oAuth2

Eine Authentifizierung mittels oAuth2 soll über Google und GitHub möglich sein. Die von Google oder GitHub zurückgelieferten Daten sollen in der Datenbank abgespeichert werden. Darüber hinaus müsste bei dem Vorhandensein spezifischer Daten, wie beispielsweise die einer E-Mail, eine automatische Zuweisung spezieller Datenbankfelder des Nutzers erfolgen.

3.2.3 Anmeldung mittels Passwortes

Für eine Anmeldung mittels Passwortes muss zuerst eine Möglichkeit verfügbar sein ein Konto zu erstellen. Bei der Erstellung eines Kontos sollten vier Felder geben sein. Das Erste für den Benutzernamen, das Zweite für die E-Mail, das Dritte und Vierte für das Passwort und die Passwort Bestätigung. Sobald der Benutzer seine Daten erfolgreich abgeschickt hat, würde das vom Client als Klartext verschickte Passwort auf dem Server verschlüsselt werden. Nachdem der Benutzer sein Konto erstellt hat, muss eine Bestätigungsmail an die angegebene E-Mail gesendet werden. Diese Bestätigungsmail sollte einen Hyperlink beinhalten mit dem das vom Nutzer erstellte Konto bestätigt werden kann. Empfängt diese E-Mail den Nutzer nicht, muss die Möglichkeit bestehen, eine erneute Bestätigungsmail zu versenden. Erst nachdem das Konto bestätigt wurde, soll es dem Nutzer gestattet sein, dieses Konto zu verwenden. Falls ein Nutzer sein Passwort vergessen hat, muss es zusätzlich eine Funktion zum Passwort wiederherstellen geben.

3.2.4 Authentifizierung nach Anmeldung

Um vom Server als angemeldeter Benutzer authentifiziert zu werden, müssen die Benutzerdaten in einem JWT gespeichert werden. Dieser JWT wird bei der Anmeldung eines Benutzers an den Client übermittelt. Ab dem Zeitpunkt wird dieser JWT bei jeder Anfrage an den Server mit gesendet. Sobald eine Anfrage den Server erreicht, muss zwangsläufig der JWT auf seine Gültigkeit geprüft werden. Hat dieser JWT eine nicht gültige Signatur oder ist bereits abgelaufen, darf keine weitere Aktion auf dem Server erfolgen.

3.2.5 Sicherheit und Login

Die Einstellungen im Bereich Sicherheit und Login sollen einem Benutzer erlauben sein bereits erstelltes Konto mit weiteren Konten zu verknüpfen. Hierbei muss der Benutzer eingeloggt sein und einen bestimmten Provider auf seiner Einstellungsseite auswählen können. Nachdem sich der Benutzer bei dem ausgewählten Provider authentifiziert hat, sollte dieses Konto dem Nutzer hinzugefügt werden. Ebenso müssen die Benutzereinstellungen eine Verwaltung der verknüpften Konten beinhalten. Das bedeutet der Benutzer kann zu jeder Zeit entscheiden, welche dieser verknüpften Konten beständig bleiben oder welche gelöscht werden.

Hat sich ein Benutzer mittels Passwortes auf Blattwerkzeug registriert, muss es für diesen Nutzer eine Möglichkeit geben sein Passwort zu ändern, selbst wenn dieser bereits

sein Konto zusätzlich mit Google oder GitHub verknüpft hat. Besteht für den Benutzer bereits ein vorhandenes Konto mit einem Passwort, sollte das neu zu verknüpfende Konto automatisch das Passwort des bereits Vorhandenen annehmen. Falls das zu verknüpfende Konto das erste mit einem Passwort sein sollte, muss dafür in den Benutzereinstellungen eine extra Passworteingabe dargestellt werden, in die das zu verwendende Passwort eingegeben wird.

Da es in Blattwerkzeug bei der Benutzernamensgebung zum jetzigen Stand keine einmaligen Benutzernamen gibt, muss es dem Benutzer in den Benutzereinstellungen zusätzlich möglich sein, seinen Benutzernamen zu ändern.

3.2.6 Rollen und Autorisierung

Die Rollen müssen sich in globale und Ressourcen spezifische Rollen unterteilen. Dabei sind Globale-Rollen wie in Sektion 4.1.4 beschrieben, keiner spezifischen Ressource zugewiesen. Zwei Beispiele globaler Rollen wären „user“ und „admin“ oder „guest“, „user“ und „admin“. Ressourcen spezifische Rollen beziehen sich zum Beispiel auf ein Projekt. Bei der Erstellung eines Projektes muss ein Benutzer eine Rolle oder eine Datenbank-Beziehung zu dem jeweiligen Projekt erhalten. Mit dieser Rolle ist es dann dem Benutzer möglich, sein Projekt zu bearbeiten oder zu löschen. Zusätzlich muss die Möglichkeit gegeben sein, einem anderen Benutzer eine Rolle zuzuweisen mit der das Bearbeiten eines von ihm nicht erstellten Projektes ermöglicht wird. Administratoren mit der „admin“ Rolle sollten jedoch Zugriff auf jedes Projekt haben.

Auflistung von nötigen Rollen und Rechten

Für eine Autorisierung mittels Rollen müssen bestimmte Regeln für die jeweiligen Controller Funktionen festgelegt werden. Diese Regeln werden mittels Pundit erstellt werden. Innerhalb dieser Regeln sollte die Überprüfung der Rollen des angemeldeten Nutzers stattfinden.

3.2.7 Bedienelemente und Routen

Die Bedienelemente, die ein Benutzer zu sehen hat, müssen jeweils von den Rollen abhängen. Dazu ist es vonnöten, bei jedem Bedienelement den Server nach der Berechtigung zu fragen oder jedoch Clientseitig Pundit nachzubauen. Besucht ein Benutzer eine Route mit nicht ausreichender Berechtigung, wird ihm der Zugriff verwehrt. Die Bedienelemente sollten benuzterfreundlich sein, da es sich bei Blattwerkzeug, wie in Sektion

2.1 beschrieben, um ein Werkzeug zu dem Lernen bestimmter Bereiche der Informatik handelt.

Code-Beispiel für Verwendung als Angular-Komponente (Template)

4 Implementierung

In dieser Sektion wird sich ausgiebig mit der Implementierung der zuvor festgelegten Anforderungen auseinandergesetzt.

4.1 Server

Da zu diesem Zeitpunkt bereits Librarys zur Authentifizierung existieren, wurde sich zuerst mit der Library Devise auseinandergesetzt.

4.1.1 Devise

Devise ist eine Ruby on Rails Library mit der sich eine flexible Authentifizierung ermöglichen lässt. Es bietet ein vorgefertigtes Datenbankschema für registrierte Benutzer, einen leichten Umgang mit OmniAuth und Funktionen wie beispielsweise das Senden einer E-Mail bei Registrierung. Im Datenbankschema enthalten ist ein Log-System mit dem beispielweise der Zeitpunkt der letzten Anmeldung eines Benutzers festgehalten werden kann.

In der Einarbeitungsphase von Devise wurde schreiben

4.1.2 Omniauth

Omniauth ist eine quelloffene Library für Ruby on Rails und ermöglicht einem, eine Anmeldung mittels unterschiedlicher Anbieter über oAuth2. Bei der Anmeldung mittels oAuth2 werden bereits viele Funktionen von Omniauth selber übernommen. Sobald der Nutzer sich bei dem jeweiligen Anbieter angemeldet hat, wird die Antwort des jeweiligen Anbieters automatisch über die von Omniauth festgelegte Route verarbeitet. Jedoch muss vorher das spezifische Gem des Anbieters für Omniauth installiert werden. Hierbei stellt jedes Gem eine eigene Strategie für Omniauth bereit. Eine Strategie stellt die Möglichkeit bereit sich mit einem speziellen Provider zu authentifizieren.

Omniauth selber verfügt nur über die Developer Strategie, diese ermöglicht eine Anmeldung ohne spezifische Überprüfung der angegebenen Daten. Das hat zur Folge, dass diese Art von Anmeldung auf keinen Fall im Produktiv System vorhanden sein darf.

Den Vorteil den Omniauth bietet ist die Kapselung zwischen den spezifischen Providern und der Hauptfunktionalität von Omniauth. Dies hat zur Folge, dass der Server nur explizit mit den installierten Providern kommunizieren kann. AuSSerdem bietet Omniauth eine lange Liste an zu installierenden Providern.

Beispiele an zu installierenden Providern:

1. GitHub[1]
2. GitLab[2]
3. Goodreads[3]
4. Google[4]

4.1.3 Pundit

Pundit ist eine Ruby on Rails Library die ein Designpattern zur Autorisierung bietet. Bei diesem Pattern wird zu einem jeweiligen Controller eine Policy angelegt. Eine Policy ist hierbei nur eine Klasse. Dabei setzt sich der Name der Policy, aus dem Namen des Models und dem Schlüsselwort Policy als Suffix zusammen. Dem Konstruktor der Policy wird beispielsweise ein Nutzer und das jeweilige Objekt übergeben, welches auf den Zugriff geprüft werden soll. Innerhalb der Policy werden die jeweiligen Controllerfunktionsköpfe in denen eine Autorisierung stattfinden soll mit einem „?“ als Suffix ergänzt und definiert. Diese Funktionen müssen zwingend einen Boolean als Rückgabewert haben um eine gültige Auswirkung als Policy zu haben. Sobald die aufgerufene Funktion der Policy fehlschlägt wird eine Exception geworfen. Diese Exception kann an jeweiliger Position beispielsweise im Controller abgefangen und verarbeitet werden.

Da es sich bei Policies um Klassen handelt, können diese auch instanziiert und jeweilige Funktionen dynamisch abgerufen werden. Dies hat zur Folge, dass explizit nach einer bestimmten Policy-Funktion gefragt werden kann, selbst wenn der Funktionsname nicht dem der aufgerufenen Policy-Funktion entspricht.

[1] https://github.com/omniauth/omniauth-github
[2] https://github.com/linchus/omniauth-gitlab
[3] https://github.com/sandboxws/omniauth-goodreads
[4] https://github.com/Yesware/omniauth-google

4.1.4 Rolify

Rolify ist eine Ruby on Rails Library zur Verwaltung von Rollen. Hierbei liefert Rolify bereits zwei Datenbanktabellen im Design der polymorphen Assoziation (Abbildung 10). Bei einer Eins-zu-viele-Assoziation hat beispielsweise ein Nutzer verschiedene Rollen, diese Rollen beinhalten verschiedene Fremdschlüssel aus verschiedenen Tabellen. Dabei ergibt sich das Problem, dass nicht mehr sicher gestellt werden kann aus welcher Tabelle der Fremdschlüssel stammt. Um dieses Problem zu lösen gibt es drei bewährte Methoden. In dieser Thesis gehen wir jedoch nur auf die von Rolify mitgelieferte Methode ein.

Bei dieser Methode handelt es sich um eine Kindtabelle „roles“ und einer Elterntabelle „users_roles“. Dabei stehen in der Roles-Tabelle die jeweiligen Informationen der Rolle und auf welche Ressource diese Rolle sich bezieht. Rolify unterscheidet hierbei zwischen globalen und Ressourcen spezifische Rollen. Eine globale Rolle beinhaltet keine Informationen einer Ressource und kann somit als beispielsweise allgemeine „user“ Rolle dienen.

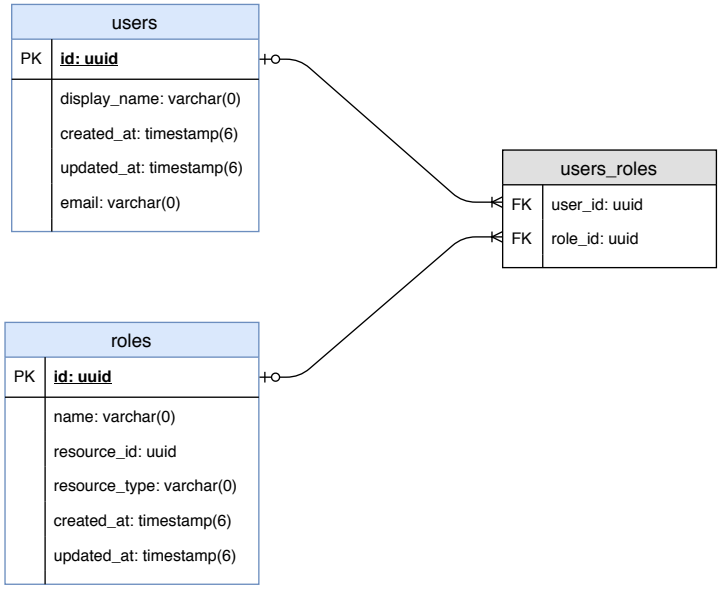


Abbildung 10: Rolifys Datenbanktabellen mit Beziehung zur users Tabelle

Die „users_roles“ beinhaltet wiederum die jeweilige „user_id“ und die zu dem Nutzer dazugehörigen Primärschlüssel einer Rolle. Dabei können verschiedene Nutzer dieselbe Rolle und ein Nutzer verschiedene Rollen haben.

AuSSerdem liefert Rolify bereits vordefinierte Funktionen mit denen es möglich ist, jeweilige Rollen eines Nutzers abzufragen, hinzuzufügen oder zu entfernen.

4.1.5 Anmeldung eines Benutzers

Die Anmeldung mittels oAuth2 oder Passwort wurde mit Omniauth ermöglicht. Hierbei wurde zuerst Omniauth selber dem derzeitigen Projekt hinzugefügt und darauffolgend die Gems zur Authentifizierung mittels Google, GitHub und Passwort. Um die einzelnen Provider nutzen zu können müssen diese in einem initializer (Listing 5) deklariert werden. Ein initializer wird nach dem Rails Framework und den dazugehörigen Gems geladen. Bei der Deklarierung der Provider ist zu beachten, dass Provider wie Google oder GitHub eine jeweilige ID und einen Secret-Key als Parameter benötigen. Diese werden jeweils auf den offiziellen Seiten der Provider erstellt. Dabei musste darauf geachtet werden, dass diese ID und dieser Secret-key über eine Umgebungsvariable mit in das Projekt eingebunden wird. Umgebungsvariablen werden in dem Fall in der Kommandozeile vor das Kommando zum starten des Projektes gesetzt. Nach dem das Kommando zum starten des Servers ausgeführt wurde, kann auf die Werte der zuvor festgelegten Umgebungsvariablen zugegriffen werden. Die Verwendung von Umgebungsvariablen ist besonders bei Blattwerkzeug vonnöten, da es sich um ein quelloffenes Projekt handelt und jeder den bereits produzierten Code einsehen kann. Da es sich bei Secret-Keys um Passwörter handelt, sind diese nicht im Repository angegeben.

```
1 Rails.application.config.middleware.use OmniAuth::Builder do
2     provider :identity, :on_registration => AuthController.action(:register),
3                         :on_login => AuthController.action(:login_with_password), :model => PasswordIdentity
4     provider :developer, :fields => [:name, :email], :uid_field => :email unless Rails.env.production?
5     provider :google_oauth2, GOOGLE_ID, GOOGLE_SECRET
6     provider :github, GITHUB_ID, GITHUB_ID
7 end
```

Listing 5: Zu Omniauth hinzugefügte Provider

Bei der Erstellung des User und des Identity Models wurde zuerst diskutiert welche Attribute die jeweiligen Models haben sollen. Hierbei steht das Identity Model für alle hinzugefügten Authentifikations-Möglichkeiten eines Benutzers. Dabei wurde darauf geachtet, welche Attribute später dem Client zur Verfügung gestellt werden sollten und welche für spätere Controller Funktionen benötigt werden.

Attribute der identities Tabelle

Die nachfolgenden Beschreibungen der Attribute beziehen sich auf die identities Tabelle, welche in Abbildung 11 dargestellt wird.

uid

Die uid ist ein String der zur eindeutigen Identifikation des Kontos eines Providers dient. Dabei kann die uid beispielsweise eine Zahl oder eine E-Mail sein. Mit der uid wird festgestellt ob ein bestimmtes Konto bereits mit einem Benutzer verknüpft ist.

provider

Das provider Attribut wird mit dem jeweiligen Provider-Namen beschrieben. Dabei sind die Namen wie in Listing 5 definiert. Da die Namen jedoch von den installierten Gems definiert werden, ist eine Änderung dieser Namen nicht ohne weiteres Möglich.

provider_data

Das provider_data Attribut wird mit jeglichen Daten beschrieben, die ein Provider über den authentifizierten Benutzer zurückliefert. (Listing 6 & 7)

```
{
  "name": null,
  "urls": {
    "Blog": "blattwerkzeug.de",
    "GitHub": "https://github.com/x"
  },
  "email": null,
  "image": null,
  "nickname": "blattwerkzeug"
}
```

Listing 6: GitHubs oAuth Daten

```
{
  "name": "Tom",
  "email": "tom@web.de",
  "image": null,
  "last_name": "Tom",
  "first_name": "Hilge",
  "email_verified": true,
  "unverified_email": "tom@web.de"
}
```

Listing 7: Googles oAuth Daten

own_data

Das own_data Attribut wird mit den Daten beschrieben die Blattwerkzeug für den Benutzer vorgesehen hat. Dies kann beispielsweise der Token zur Aktivierung einer E-Mail sein.

type

Das type Attribut wird, sobald es der Tabelle hinzugefügt wurde, von Rails automatisch interpretiert. Hierbei handelt es sich um Single Table Inheritance (STI). STI ist eine Möglichkeit Objekt-Orientierung in einer Relationellen Datenbank zu emulieren. Dabei ist die Tabelle identities und dessen Model (Identity) die Basisklasse und die in type festgelegten Klassen, die Abgeleiteten. Das type Attribut

ermöglicht einen sofortigen Zugriff auf die Klasse des Providers einer ausgewählten Identity.

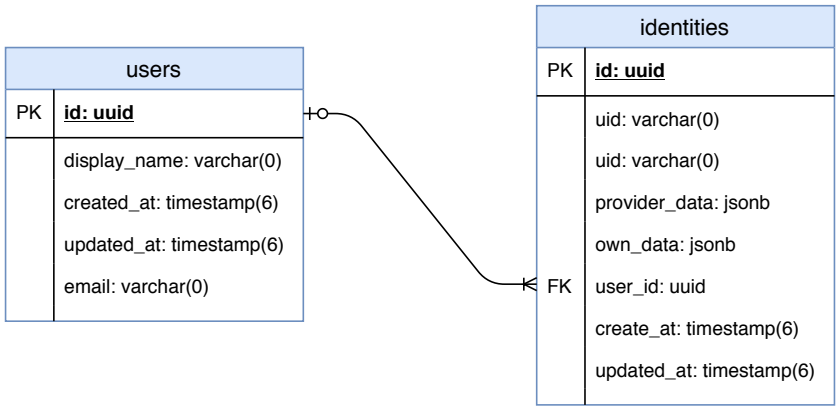


Abbildung 11: Darstellung der users und identities Tabelle inklusive Beziehung.

Attribute der users Tabelle

Die nachfolgenden Beschreibungen der Attribute beziehen sich auf die users Tabelle, welche in Abbildung 11 dargestellt wird.

display_name

Das display_name Attribut steht für den Anzeigenamen des Benutzers. Dieser ist jedoch nicht einzigartig und lässt sich beliebig oft verändern. Das hat zur Folge, dass mehrere Benutzer denselben Anzeigenamen haben können.

email

Das email Attribut steht für die primäre E-Mail eines Benutzers. Die primäre E-Mail wird verwendet sobald eine E-Mail von Blattwerkzeug an den Benutzer gesendet werden soll. Dies ist wichtig, da jedes verknüpfte Konto eine unterschiedliche E-Mail besitzen darf. Die primäre E-Mail wird bei Erstellung eines Benutzers gesetzt, auSSer ein Benutzer authentifiziert sich über einen Provider dessen Rückgabe keine E-Mail enthält.

Anmeldemöglichkeiten

Die geforderten Anmeldemöglichkeiten und deren Implementierung werden in diesem Abschnitt erläutert.

Anmeldung mittels oAuth2

Bei einer Anmeldung mittels oAuth2 wurde bei dem Provider über den sich authentifiziert wird, eine Redirect Url hinterlegt. Auf diese wird der jeweilige Benutzer nach Authentifikation zurückgeleitet. Die Route zu der Redirect Url zeigt auf die Funktion mit dem Namen „Callback“ im „AuthController“. Die Callback Funktion prüft auf das vorhanden sein der vom Provider zurückgelieferten Daten. Sollte bisher keine Identität mit den Daten des Providers erstellt worden sein wird eine neue Identität angelegt. Sollte ein angemeldeter Benutzer diese Funktion ausführen wird ihm eine neue Identität zu seinem Benutzer hinzugefügt. Sollte es sich um keinen angemeldeten Benutzer handeln, wird zusätzlich ein neuer Benutzer erstellt. Da jeder Provider unterschiedliche Daten zurückliefert wird beim anlegen einer neuen Identität zwischen den einzelnen Providern unterschieden. Jeder Provider besitzt eine Abgeleiteteklasse der Basisklasse Identity und greift innerhalb seiner Klasse auf die für ihn relevanten Daten zu. Sollte eine Identität mit den zuürckgelieferten Daten bereits existieren, wird ohne Erstellung einer Identität fortgefahren. Mit der erstellten oder existierenden Identität werden die Benutzerinformationen in die Payload eines JWT geschrieben. Bei den Benutzerinformationen handelt es sich um die id, den display_name und die Rollen eines Benutzers. Die Funktionen zu Erstellung eines JWT wurden hierbei in einen Helper ausgelagert und die Gültigkeitsdauer des erstellten JWT beläuft sich auf eine Stunde.

Anmeldung mittels Passwort

Visualisierung als Automat (Zustände / Übergänge) zur Veranschaulichung des Prozesses vermutlich hilfreich

Allgemein: Mehr Struktur, in diesem Punkt verstecken sich zuviele eigenständige Funktionen

Die Anmeldung mittels Passwort wurde mittels Omniauth Identity[5] ermöglicht. Omniauth Identity ist ebenfalls eine Strategie mit der die Möglichkeit gegeben ist, sich über ein Passwort bei Blattwerkzeug anzumelden. Ebenso wie die Developer Strategie, bietet auch diese Strategie die Möglichkeit ein vorgefertigtes Formular für Anmeldung und Registrierung zu erstellen. Jedoch ist es bei dieser Strategie optional und eine ausschlieSSliche Kommunikation über ein API ist möglich.

[5] `https://github.com/omniauth/omniauth-identity`

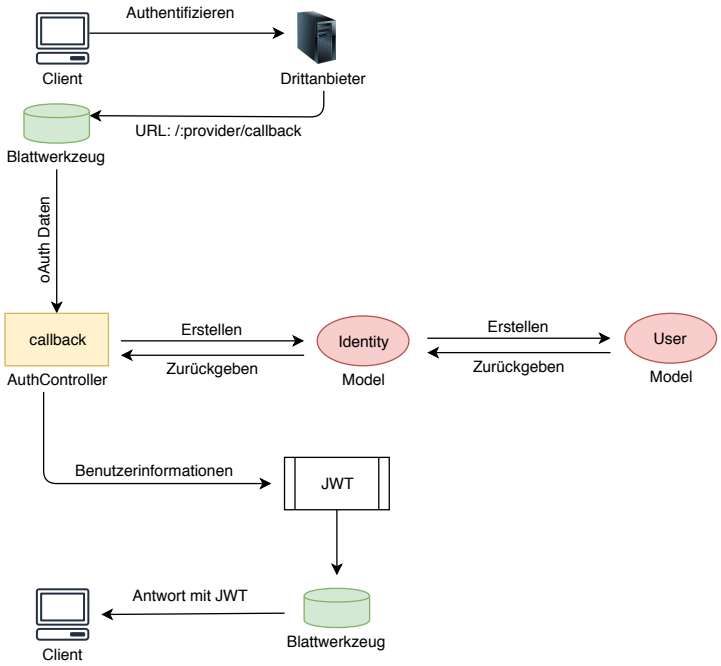


Abbildung 12: Ablauf einer Anmeldung mittels oAuth2

Bei dem Nutzen dieser Strategie fiel auf, dass die Daten übermittelt wurden sie jedoch nicht richtig verarbeitet werden konnten (Listing 8). Das hatte zur Folge, dass mit der Erstellung eines Models, welches mit als Parameter im iniziliazer angegeben werden kann, nicht ordnungsgemäSS fortgefahren werden konnte. Um fortzufahren wäre es möglich eine eigene Strategie für Omniauth zu entwickeln oder auf die von Omniauth Identity mitgelieferten Optionen zurück zugreiffen.

```
def registration_phase
    attributes = (options[:fields] + [:password, :password_confirmation])
        .inject({}){|h,k| h[k] = request[k.to_s]; h}
    @identity = model.create(attributes)
    if @identity.persisted?
        env['PATH_INFO'] = callback_path
        callback_phase
    else
        if options[:on_failed_registration]
            self.env['omniauth.identity'] = @identity
            options[:on_failed_registration].call(self.env)
        else
            registration_form
        end
    end
end
```

Listing 8: Aufgerufene Funktion bei POST Request (Omniauth Identity)

Omniauth Identity bietet die Möglichkeit bei Registrierung auf eine Funktion zu verweisen. Diese Funktion muss ebenfalls im iniziliazerangegeben werden. (Listing 5). Sollte bei der Erstellung des Models ein Fehler auftreten (Listing 8), wird die Funktion im zuvor festgelegten iniziliazer aufgerufen. Letztendlich wurde sich in der Thesis für diese Methode entschieden. Hierbei wurden jegliche Funktionen innerhalb des AuthControllers und auSSerhalb der Strategie genutzt.

typo

Bevor eine Anmeldung mittels Passwort stattfinden kann, wurde eine Registrierungs-Möglichkeit hinzugefügt. Die „register“ Funktion arbeitet ähnlich wie die call-back Funktion. Der Unterschied besteht darin, dass die register Funktion einen simulierten Hash in der Struktur Omniauths erhält und mit diesem versucht eine neue Identität zu erstellen. Beim erstellen einer neuen Password-Identität wird ein Verifizierungs-Token generiert. Dieser Verifizierungs-Token ist eine Universally Unique Identifier (UUID) und wird zur Verzifizierung der Identität genutzt. Nachdem eine Identität erstellt wurde, sendet der Server mithilfe der Basisklasse ActionMailer eine Bestätigungsmail. Diese Bestätigungsmail verhindert das Registrieren von willkürlichen E-Mail Adressen. AuSSerdem wird mit der Bestätigungsmail eine E-Mail Adresse auf ihre existens geprüft. Die Bestätigungsmail besteht aus einem Hyperlink zu einer Uniform Resource Locator (URL) Blattwerkzeugs in der, der Verifizierungs-Token enthalten ist. Sobald dieser URL besucht wird,

welche

wird die Indentität als bestätigt gekennzeichnet. Dies erfolgt durch das Setzen des confirmed Feldes im JSON Blob Attribut own_data.

Da es vorkommen kann, dass eine Verifizierungsmail nicht an der angegebene E-Mail ankommt, wurde hierfür eine Funktion im IdentitiesController geschrieben. Diese Funktion prüft die übermittelte E-Mail auf ihre existens in der Blattwerkzeug-Datenbank und ob diese bereits verifiziert wurde. Sollte diese E-Mail nicht verifiziert sein wird eine neue Verifizierungsmail versendet. Gleichzeitig wird jedoch auch eine Wartezeit von zwei Minuten im own_data Attribut festgelegt. Die Wartezeit zum versenden einer neuen Verifizierungsmail soll verhindern, dass eine E-Mail Adresse mit Verifizierungsmails überhäuft werden kann.

Sollte eine Identität mit einem Passwort bereits vorhanden sein, kann sich mit seiner E-Mail und seinem Passwort angemeldet werden. Bei der Anmeldung wird das Passwort zur angegebenen E-Mail überprüft. Zusätzlich wird geprüft ob die E-Mail verifiziert wurde. Sollten diese Sicherheitsabfragen erfolgreich durchlaufen sein wird wie bei der Anmeldung mittels oAuth2 ein JWT ausgestellt in dem die Benutzerinformationen gespeichert werden.

Da im Laufe der Zeit die Möglichkeit besteht, dass Benutzer ihr Passwort vergessen haben, wurde hierfür eine Funktion im IdentitiesController erstellt. Damit ein Passwort zurück gesetzt werden kann muss die übermittelte E-Mail Adresse als Passwort Identität vorhanden sein. Sollte diese Identität vorhanden sein, wird ein password_reset_token erstellt. Dieser password_reset_token ist eine UUID und wird im own_data JSON Blob Attribut gespeichert. Ebenfalls zum own_data Attribut hinzugefügt wird ein Feld password_reset_token_exp. Dessen Wert beläuft sich auf die aktuelle Uhrzeit plus dreiSSig Minuten. Nachdem die festgelegte Uhrzeit des password_reset_token_exp Feldes überschritten wurde, kann der Zurücksetzungs-Token nicht mehr verwendet werden. Damit das Passwort trozdessen zurückgesetzt werden kann, muss ein neuer Zurücksetzungs-Token angefordert werden. Die Ablaufzeit eines Zurücksetzungs-Tokens hat den Vorteil, dass sollte beispielsweise ein Angreiffer Zugriff auf den Browserverlauf haben, dieser nicht die Möglichkeit erhält das Passwort dieses Benutzers zu verändern. Die E-Mail die beim Kennwort zurücksetzen versendet wird, wird an die primäre E-Mail verschickt und enhält einen Hyperlink zum wiederherstellen des Passwortes. Der Hyperlink beinhaltet den password_reset_token und nachdem ein neues Passwort

ausgewählt wurde, wird von jeder Passwort-Identität des Benutzers das Passwort geändert.

Blattwerkzeug nutzt Subdomains zur Unterteilung verschiedener Sprachen. Dabei ergab sich das Problem, dass das Umleiten auf eine Basisurl die Cross-Site-Request-Forgery (CSRF) Protection von Rails ausgelöst hat. Somit ist es derzeit möglich sich über die Subdomains anzumelden, jedoch wird zwischenzeitlich ein Fehler zurückgegeben.

4.1.6 Autorisierung

Sobald ein Benutzer angemeldet ist, wird bei jedem Request ein JWT mit an den Server gesendet. Dieser wird Serverseitig auf seine Gültigkeit geprüft. Hierbei prüft der Server ob dieser JWT seine Ablaufzeit nicht überschritten hat und ob der JWT die Signatur des Servers beinhaltet. Diese Überprüfung authentifiziert einen Besucher als angemeldeten Benutzer und wird von der Library jwt[6] übernommen.

Damit zwischen den Benutzern unterschieden werden kann, wurden drei Globale-Rollen eingebaut.

Unklar: Was genau wurde extra „eingebaut“? Ich halte das eher für eine Konvention. Und zwischen Benitzern unterscheidet man anhand ihrer ID ;-) Ich weiSS was gemeint ist, aber geh hier nochmal in dich und formulier sauber aus inwiefern diese (sinnvolle!) Konvention dir hilft.

guest

Die guest Rolle wird einem einzigen Benutzer zugewiesen. Dieser Benutzer beschreibt einen unangemeldeten Besucher. Jedem der Blattwerkzeug besucht und sich nicht anmeldet wird dieser Gast-Benutzer zugewiesen. Der Gast-Benutzer besitzt keine Möglichkeit Änderungen, die über Bedienelemente ermöglicht werden, zu speichern oder zu löschen.

user

Die user Rolle stellt einen angemeldeten Benutzer dar. Sollte bei einer Identitäts Erstellung festgestellt werden, dass der aktuelle Benutzer keine Globale-Rolle admin oder user besitzt, wird automatisch die user Rolle hinzugefügt. Mit der user Rolle wird das Erstellen, Speichern und Löschen von Projekten ermöglicht. Dies bezieht sich jedoch nur auf eigene Projekte.

[6] `https://github.com/jwt/ruby-jwt`

admin

Die admin Rolle stellt einen administrierenden Benutzer dar. Jegliche Operationen die mit der user Rolle ausgeführt werden können, können ebenfalls mit der admin Rolle ausgeführt werden. Darüberhinaus kann ein Benutzer mit der admin Rolle, jegliche Projekte bearbeiten und beschränkt sich nicht auf seine eigenen. Der Admin-Benutzer besitzt zusätzliche Funktionen mit denen er beispielsweise eine Neuigkeit, die Clientseitig auf der Startseite angezeigt wird, erstellen kann.

Um den Zugriff auf das Verwalten eines Projektes oder einer News zu beschränken, wurde jeweils eine Policy (Listing 9) angelegt. Bei der Erstellung einer Policy fiel auf, dass es sinnvoller ist den Ersteller einer News oder eines Projektes in der jeweiligen Tabelle mit zu hinterlegen. Dafür wurde ein neues Attribut owner der projects und news Tabelle hinzugefügt. Dieses owner Attribut ist ein Fremdschlüssel und bezieht sicht auf die users Tabelle. Die Beziehung die folgedessen enstand ist eine 1:n. Das beudetet ein Benutzer kann beispielsweise verschiedene Projekte besitzen, jedoch ein Projekt nur einen Benutzer haben. Den Vorteil den das hinzufügen des owner Attributes hat, ist der direkte Zugriff auf die erstellten Projekte eines Benutzers. AuSSerdem kann zwischen einem Ersteller eines Projektes und einem Benutzer der eine Rolle zum bearbeiten eines Projektes erhalten hat, unterschieden werden kann.

```ruby
class ProjectPolicy
    attr_reader :user, :project

    def initialize(user, project)
        @user = user
        @project = project
    end

    def create?
        user.has_role?(:user) || user.has_role?(:admin)
    end

    def update?
        user.owner_of?(project) || user.has_role?(:project_editor, project) || user.has_role?(:admin)
    end

    def destroy?
        user.owner_of?(project) || user.has_role?(:admin)
    end
end
```

Listing 9: Policy zur Autorisierung eines Zugriffs auf ein Projekt

Da Blattwerkzeug bereits eine Passwortabfrage eingebaut hatte (Listing 10) dessen Anmeldedaten sich auf user und user beschränkten, mussten diese durch die von Pundit mitgelieferte authorize (Listing 11) Methode ersetzt werden.

```ruby
  # Update an existing project.
  def update
    ensure_write_access do
      current_project.update project_update_params # Simple properties
      current_project.update project_used_block_languages_params # Used block languages

      render json: current_project.to_project_api_response
    end
  end
```

Listing 10: Controller-Funktion ohne Integrierung Pundits

```
1   # Update an existing project.
2   def update
3     begin
4       authorize current_project
5       current_project.update project_update_params # Simple properties
6       current_project.update project_used_block_languages_params # Used block languages
7
8       render json: current_project.to_project_api_response
9     rescue Pundit::NotAuthorizedError => e
10      error_response("You need the permission")
11    end
12  end
```

Listing 11: Controller-Funktion mit Integrierung Pundits

4.1.7 may_perform

Sobald sich mit den Rollen beschäftigt wurde, stellte sich die Frage wie die Bedienelemente in Abhängigkeit von den Rollen Clientseitig angezeigt werden sollten. Hierbei gab es die Möglichkeit ein eigenes Pundit (Sektion 4.1.3) ähnliches Mustur zu implementieren oder den Server bei jedem Bedienelement zu fragen, ob der aktuelle Benutzer Zugriff auf das Bedienelement hat. In der Thesis wurde sich für den zweiten Weg entschieden, da bei einer zusätzlichen Clientseitigen Überprüfung, Client und Server gepflegt werden müssten. Sollte es vorkommen, dass die Pflege des Serverseitigen-Teils vergessen wurde, stellt dieses ein Sicherheitsrisiko dar. Clientseitige Anwendungen werden auf dem Endgerät eines Benutzers ausgeführt. Folgedessen besteht die Möglichkeit die Anwendung zu manipulieren.

Die Serverseitige Überprüfung der Bedienelemente wurde mittels der may_perform Funktion realisiert. Diese erhält vom Client eine Liste an Daten in der jedes Element ein Bedienelement darstellt. In der may_perform Funktion wird jedes Element der Liste durchlaufen und auf das Zugriffsrecht des aktuellen Benutzers geprüft. Dies geschieht in dem aus den übermittelten Daten eine Instanz einer Policy erstellt wird die zu der jeweiligen Ressource, beispielsweise der Projekte (Listing 9) gehört. Hierbei wird die Funktion die auf ihren Zugriff geprüft werden soll, ebenfalls vom Client mit an den Server gesendet. Diese wird schlussendlich auf der erstellten Instanz ausgeführt. Das Ergebnis der Aufgerufenen Policy Funktion wird in einem Array gespeichert und sobald die Liste durchlaufen wurde an den Client übermittelt. Die Nutzung des Arrays und der übermittelten Liste wird in der Client Sektion 4.2 erläutert.

4.1.8 Sicherheit und Login

Da ein Benutzer die Möglichkeit haben soll seinen Account zu verwalten und Änderung an diesem vorzunehmen, wurden fünf Einstellungsmöglichkeiten implementiert. Damit ein Benutzer Einstellungen an seinem Account vornehem kann, muss dieser sich vorerst anmelden.

Konto verknüpfen

Die callback Funktion im Authcontroller enthält eine Abfrage zur Überprüfung eines angemeldeten Benutzers. Bei einem angemeldeten Benutzer wird die zuerstellende Identität dem angemeldeten Account hinzugefügt. Dies wird mittels eines Fremdschlüssels, der auf die id eines Benutzers referenziert, realisiert.

Konto löschen

Damit eine Identität gelöscht werden kann müssen zuerst einige Sicherheitsabfragen durchlaufen werden. Die E-Mail der zu löschenden Identität darf nicht als derzeitige primäre E-Mail fungieren. Folgedessen kann beim einem Fremdzugriff auf den Account, keine Übernahme des Benutzers erfolgen. Ebenfalls muss eine weitere bestätigte Identität vorhanden sein. Demnach ist es nicht möglich jegliche Authentifizierungs Möglichkeiten eines Benutzers zu entfernen. Um zu verhindern, dass mit verfälschten Daten fremde Identitäten gelöscht werden, muss die zu löschende Identität zwangsläufig dem angemeldeten Benutzer angehören.

Primäre E-Mail wechseln

Ein Benutzer hat die Möglichkeit die primäre E-Mail zu ändern. Hierbei wurde darauf geachtet, dass die Änderung einer primären E-Mail ebenfalls bestätigt werden muss. Das Bestätigen einer Änderung der primären E-Mail schützt vor Benutzer übernahmen innerhalb Blattwerkzeugs. Bei einer Änderung der primären E-Mail wird eine Bestätigungsmail vom Server gesendet, allerdings wird vorher überprüft ob die neue E-Mail in einem der verknüpften Identitäten vorhanden ist. Folglich ist es nicht Möglich mit manipulierten Daten, auf eine nicht verknüpfte und nicht bestätigte E-Mail zu wechseln. Die Bestätigungsmail enthält einen Hyperlink mit einem zuvor erstellten change_primary_email_token. Damit die neue primäre E-Mail nicht als beispielsweise URL-Parameter mit an die URL gehängt werden muss, wurde der change_primary_email_token in der Identität der neuen primären E-Mail gespeichert. Demnach kann mittels Token direkt auf die wechselnde Identität zugegriffen werden. Für eine erfolgreiche Änderung der primären E-Mail, muss der

```
1  validates_format_of :display_name, :with => /\A[a-zA-Z0-9]{3}.{0,17}\z/i
```

Listing 12: Validierung des Benutzernamens

an die URL angehängte Token dem change_primary_email_token entsprechen. Zusätzlich darf der angegebene Token seine Ablaufzeit nicht überschritten haben. Sollte die vom Benutzer ausgewählte E-Mail bereits als primäre E-Mail eines anderen Benutzers dienen, kann der Vorgang zur Änderung ebenfalls nicht erfolgreich abgeschlossen werden.

Passwort wechseln

Damit ein Passwort wechsel durchgeführt werden kann, muss eine Passwort Identität vorhanden sein. Um das Passwort der Passwort Identität zu verändern muss das derzeitige Passwort und eine neues Passwort, an den Server übermittelt werden. Der Server gleicht das übermittelte Passwort mit dem Passwort der verknüpften Identität ab. Sollten bereits mehrere Passwort Identitäten existieren, wird von jeder das Passwort geändert. Da bei einer Verknüpfung einer Passwort Identität, das Passwort einer bereits existierenden Passwort Identität übernommen wird, führt das ändern jeglicher Passwort Identitäten zu keinem ungewollten verhalten.

Benutzernamen wechseln

Da es in Blattwerkzeug keine eindeutigen Benutzernamen gibt, wurde eine Funktion zum wechseln des Benutzernamens hinzugefügt. Diese Funktion prüft lediglich auf das Valide sein eines Benutzers nach Änderung des Benutzernamens. Zur Überprüfung des Benutzernamens wurde ein Validator hinzugefügt, der den Benutzernamen mittels Regulären Ausdrucks auf seine Gültigkeit prüft. (Listing 12) Der Benutzername muss mit drei Buchstaben oder Zahlen beginnen und kann gefolgt werden von siebzehn Zeichen.

Die Regex-Regel ist so nicht sinnvoll. Webseiten beschränken Benutzernamen vor allem um Probleme mit leicht zu verwechselnden Zeichen zu vermeiden (Stichwort „Greek Question Mark Prank“) oder weil Namen wie „<script>alert(haha)</script>“ gar nicht erst möglich sein sollen. Beschreibe beide Sachverhalte / Probleme (nicht unbedingt hier, das kann sogar bis zu den Anforderungen nach vorne) und präsentiere hier dann einen passenderen RegEx.

4.2 Client

Die Client Sektion beschreibt die Umsetzung der Anforderungen (clientseitig).

4.2.1 Nebular

Nebular ist eine Angular Library die bereits viele Bedienelemente im benutzerfreundlichen Design enthält. Zusätzlich stellt Nebular bereits Services und Helper Koponenten mit denen der Umgang mit oAuth2 oder JWT erleichtert wird. Da Nebular jedoch eine so umfangreiche Libraby ist und wir im Fall dieser Thesis nur auf die Struktur und das Design der Bedienelemente zurückgegriffen hätten, wäre eine Vielzahl an zusätzlichen Eigenschaften Nebulas ungenutzt. Zusätzlich konzentriert sich Nebular bei Autorisierung stark auf den Clientseitigen Teil. Dies hat zur Folge, dass serverseitig und clientseitig Fehler abgefangen werden müssten. Aus diesen Gründen wurde sich gegen Nebular und für ein eigenes Design entschieden.

4.2.2 Dialog zur Anmeldung/Registrierung

Zuerst wurde sich mit der Darstellung von Komponenten für die Anmeldung und die Registrierung auseinandergesetzt. Diese sollten einem neumodischen Stil entsprechen und leicht zu bedienen sein. Hierbei bietet Angular Material eine Möglichkeit, sogenannte Dialog-Fenster zu erstellen. Um das Dialog-Fenster zu implementieren muss ein Modul namens „MatDialogModule“ eingebunden werden. Angular Material bietet auSSerdem die Möglichkeit sogennante „tabs“ zu verwenden. Diese Tabs ermöglichen den wechsel zwischen unterschiedlichem Inhalt mit zusätzlicher Animation und gleichbleibender Komponente. Aus diesen beiden Elementen von Angular Material wurde schluessendlich ein Popup-Fenster erstellt in dem der Wechsel zwischen Anmeldung (Abbildung 13a) und Registrierung (Abbildung 13b) ermöglicht wird.

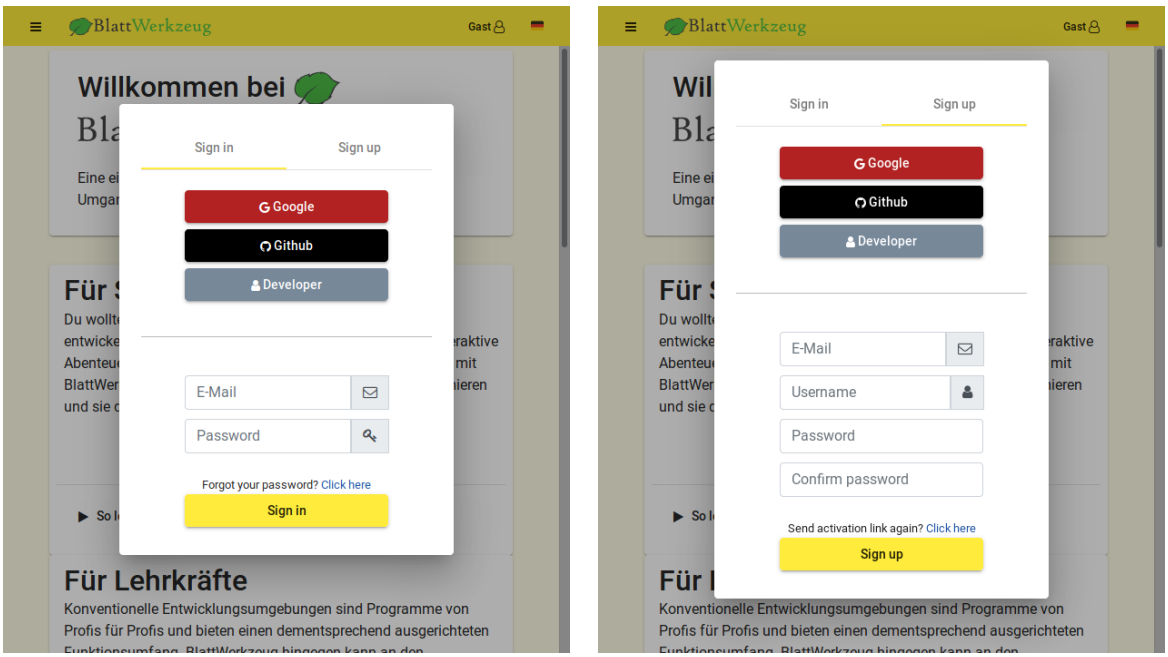
ugs

Dieser Satz beschreibt Anforderungen

Kommunikation mit dem Server

Die Kommunikation mit dem Server findet mittels HTTP-Anfragen statt. Um eine HTTP-Anfrage zu ermöglichen, wird das HTTP-Modul von Angular verwedent.

Während der Arbeit mit Blattwerkzeug, wird ein Entwickler zwangsläufig mit dem Caching-System konfrontiert. Dieses speichert den zuletzt erhaltenen Wert einer HTTP-Anfrage. Sollte auf das Observable, welches für die HTTP-Anfrage zuständig ist, erneut zugegriffen werden, wird der gespeicherte Wert zurückgegeben. Ein



(a) Darstellung eines Dialogs zur Anmeldung (b) Darstellung eines Dialogs zur Registrierung

Abbildung 13: Dialog zur Anmeldung/Registrierung

gespeicherter Wert wird erst verändert, sobald explizit eine neue Anfrage an den Server gestellt wird.

Für die Kommunikation mit dem Server wurde mit dem bereitgestellten Caching-System gearbeitet. Damit die zu übermittelnden Daten einer Datenstruktur zugeordnet werden können, wurden Interfaces erstellt. Das Verwenden eines Interfaces als Datenstruktur erleichtert die Identifikation der Zugehörigkeiten der zu übermittelnden Daten.

Provider-Buttons

Damit zwischen Produktiv-, Entwicklungs-, und Testumgebung unterschieden werden kann, erhalten die Provider Buttons ihre Informationen mittels get Anfragemethode vom Server. Dies bietet den Vorteil, dass Serverseitig definiert werden kann, welche Provider dem Benutzer zur Authentifizierung bereit gestellt werden sollen.

Die Darstellung der Provider-Buttons wird in zwei Komponenten unterteilt. Die provider-button Komponente dient der Darstellung des einzelnen Buttons. Diese enthält jegliche Informationen über den Provider und die Darstellung des Buttons.

Die provider-all-buttons Komponente zeigt restlos jeden derzeitig verfügbaren Provider an. Innerhalb der provider-all-buttons Komponente wird auf die provider-button Komponenten zurückgegriffen.

Validate-Input

Der Auth-Dialog beinhaltet verschiedene Input-Felder. Um eine Rückmeldung bei Fehleingabe in einem der Input-Felder zu erhalten, muss ein Input-Feld validiert werden. Hierbei sind verschiedene Möglichkeiten gegeben.

Die Validierung eines Input-Feldes ist ein fester bestandteilvon HTML5. Für eine Validierung mittels HTML5 Validator, muss der Validator als Attribut dem Input-Feld hinzugefügt werden.

typo

Die Angular Validierung bietet bereits vordefinierte Klassenmethoden zur Validierung von Input-Feldern. Die vordefinierten Klassenmethoden gleichen, von der Funktionsweise, den HTML Validatoren. Ein Vorteil den Angular hierbei bietet, ist das Erstellen eigener komplexer Validatoren. Ein weiterer Vorteil, ist das Auslagern der Validatoren aus dem Template. Folglich kann beispielsweise ein Service zur Validierung von Input-Feldern dienen und somit anwendungsübergreifend die Validierung geändert werden.

Für die Entscheidung zwischen HTML5 und Angular Validierung, musste abgewägt werden in welchem Umfang die Validierung benötigt wird. Die zu Nutzenden Input-Felder können jeweils mit den vordefinierten Validatoren, beider Validierungs-Möglichkeiten, validiert werden. Aus diesem Grund wurde sich für die HTML5 Validierung entschieden, da diese über das Hinzufügen als Attribut eine deutlich komfortablere Lösung bietet.

Bei der Implementierung der jeweiligen Validatoren innerhalb der Input-Felder wurde erhöhte Code-Redundanz festgestellt. Aus diesem Grund wurde für das Validieren der einzelnen Input-Felder eine validate-input Komponente erstellt. Die validate-input Komponente lädt dynamisch ein Input-Feld mittels ng-content in das Template. Der zu validierende Wert, wird mittels input-binding an die validate-input Komponente übermittelt (Listing 13). Zusätzlich kann mit input-binding das Design des Input-Feldes (Abbildung 14) erweitert werden. Sollte die Validierung des übermittelten Wertes fehlschlagen, wird undefined an die Eltern-Komponente zurückgeliefert. Dies hat den Vorteil, dass der Client von einem invaliden Wert ausgehen kann und der Server keine invaliden Werte, auSSer undefined, erhält.

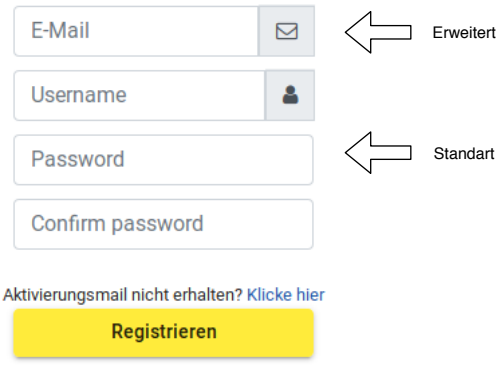


Abbildung 14: Standart und erweiterte Ausführung eines validate-input

Allerdings verhindert diese Komponente nur bei falscher Eingabe das Übermitteln von falschen Daten. Aus diesem Grund ist diese Komponente kein Ersatz für die serverseitge Validierung.

Standard! Und verpass den Icons mal die CSS-Klasse fa-fw, dann werden die identisch breit dargestellt.

```
1  <validate-input
2      icon="key"
3      [(value)]="signInData.password">
4
5      <input #inputRef
6        type="password"
7        class="form-control"
8        placeholder="Password"
9        required>
10 </validate-input>
```

Listing 13: Verwendung einer validate-input Komponente

Inspiration

4.2.3 Speichern eines JWT

Bei dem erstellen eines JWT muss abegewägt werden wo dieser Clientseitig gespeichert werden soll. Hierbei gibt es die Möglichkeit den JWT im LocalStorage, sessionStorage oder als Cookie zu speichern.

Der LocalStorage und der sessionStorage sind ähnlich und bieten dieselben schwachstellen, weSShalb sich im weiteren Verlauf nur auf den LocalStorage bezogen wird. Bei der Speicherung eines JWT im LocalStorage wird Cross-Site-Scripting (XSS) zur Schwachstelle. Bei XSS handelt es sich um mit einer der bekanntesten Angriffsmethoden in der Webentwicklung. Hierbei wird beispielsweise ein Hyperlink manipuliert und mit JavaScriptauf den localStorage zugegriffen. Sobald sich ein Angreifer den JWT zukommen lassen hat, ist es für diesen möglich sich mit dem JWT zu authentifizieren. Angular schützt seine Applikationen bereits vor XSS in dem beispielsweise Eigenschaften und Attribute auf nicht vertrauenswürdige Werte überprüft werden.

Die Speicherung im Cookie bietet die Möglichkeit, mit dem httpOnly-Flag, den Zugriff mittels JavaScriptzu verbieten. Zusätzlich kann der secure-Flag gesetzt werden, der die Übertragung des Cookies nur über eine sichere Hypertext Transfer Protocol Secure (HTTPS) Verbindung zulässt. Jedoch hat die Speicherung eines Cookies ebenfalls eine groSSe Schwachstelle. CSRF ist ebenso berühmt in der Webentwicklung wie XSS. Bei CSRF handelt es sich um das fälschen von Anfragen einer Webseite auf die kein Einfluss genommen werden kann. Dabei wird die Anfrage so gefälscht, dass der Server von einem legitimen eingeloggten Benutzer ausgeht.

Bei der Entscheidung zwischen XSS oder CSRF muss abgewägt werden, was einem sicherer erscheint. Da sich bei der Verhinderung von XSS sehr stark auf Angular verlassen wird, wurde sich für CSRF entschieden. Das bedeutet, der JWT wird in einem Cookie mit dem httpOnly- und dem secure-Flag versendet.

4.2.4 Sicherheit und Login

Die clientseitige Einstellungsmöglichkeit für Sicherheit und Login, kann erst aufgerufen werden, sobald ein Benutzer sich angemeldet hat. Für die Input-Felder wurde die bereits erstellte validate-input Komponente verwendet. Damit zwischen den Einstellungsmöglichkeiten unterschieden werden kann, wurde jeweils eine Komponente erstellt. Da die Einstellungsmöglichkeiten sich auf ein paar wenige einschränken, wurden die erstellten Komponenten in einer Komponente zusammen getragen.

Passwort ändern

Sollte ein Benutzer sein Passwort ändern wollen, muss dafür vorerst eine Passwort Identität vorhanden sein. Falls keine Passwort Identität vorhaden ist, wird dem angemeldeten Benutzer keine Passwort Änderung angezeigt. Damit ein Benutzer nicht versehentlich ein falsches Passwort angibt, muss das eingegebene Passwort

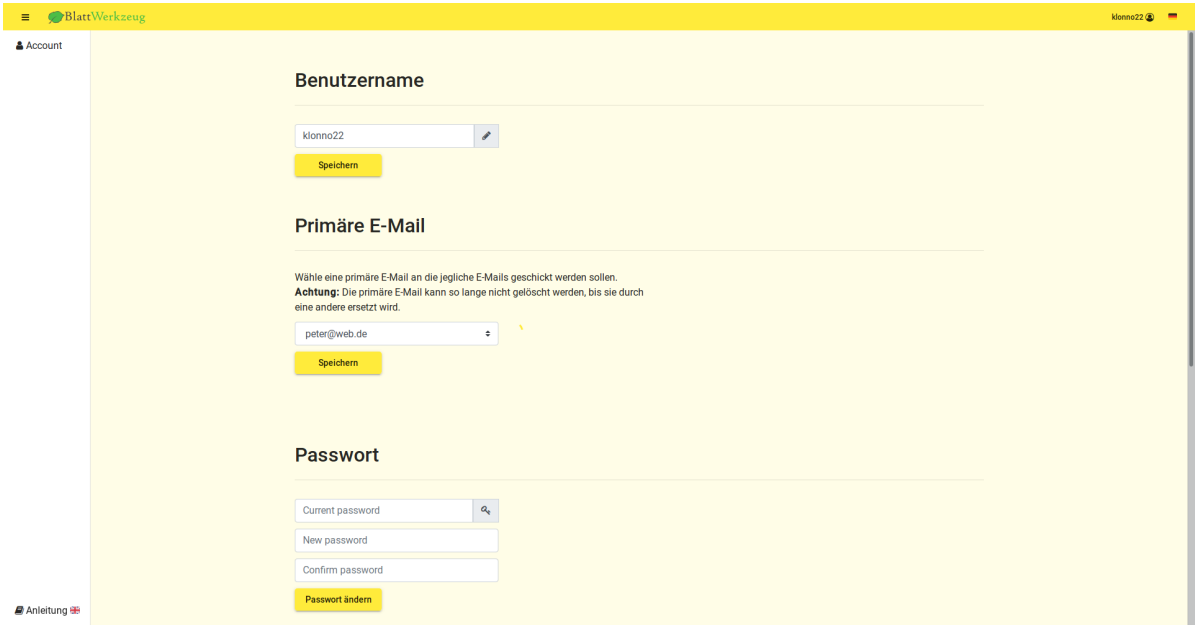


Abbildung 15: Standart und erweiterte Ausführung eines validate-input

bestätigt werden.(Abbildung 15) Hierfür muss das neue Passwort erneut eingegeben werden. Damit ein Passwortwechsel erfolgreich durchgeführt werden kann, muss der Server die Eingaben auf ihre Gültigkeit prüfen. (Sektion 4.1.8)

Verknüpfte Konten

Ein Benutzer soll die Möglichkeit besitzen seine bereits verknüpften Konten zu verwalten. Hierzu wurde eine Übersicht der bereits verknüpften Konten erstellt (Abbildung 16). Diese Übersicht der Konten bietet die Möglichkeit, die bereits verknüpften Konten zu entfernen oder Profilseiten, falls vom Provider ausgeliefert, zu besuchen. Die Entscheidung ob ein verknüpftes Konto entfernt werden darf, obliegt jedoch dem Server. (Sektion 4.1.8)

Konto Verknüpfung

Damit ein angemeldeter Benutzer sein bereits bestehendes Konto mit weiteren Konto verknüpfen kann wurde die bereits erstellte provider-button Komponenten zur Darstellung verwendet. Hierbei ist die Funktionsweise in den Einstellungen identisch zu der Funktionsweise des Dialogs (Sektion 4.2.2).

4.2.5 Wrapper-Komponenten

Die Darstellung der Blattwerkzeug-Seite muss abhängig von den Rollen eines Benutzers sein. Hierfür ist es möglich die Unterteilung der einzelnen Codeabschnitte jeweils mit

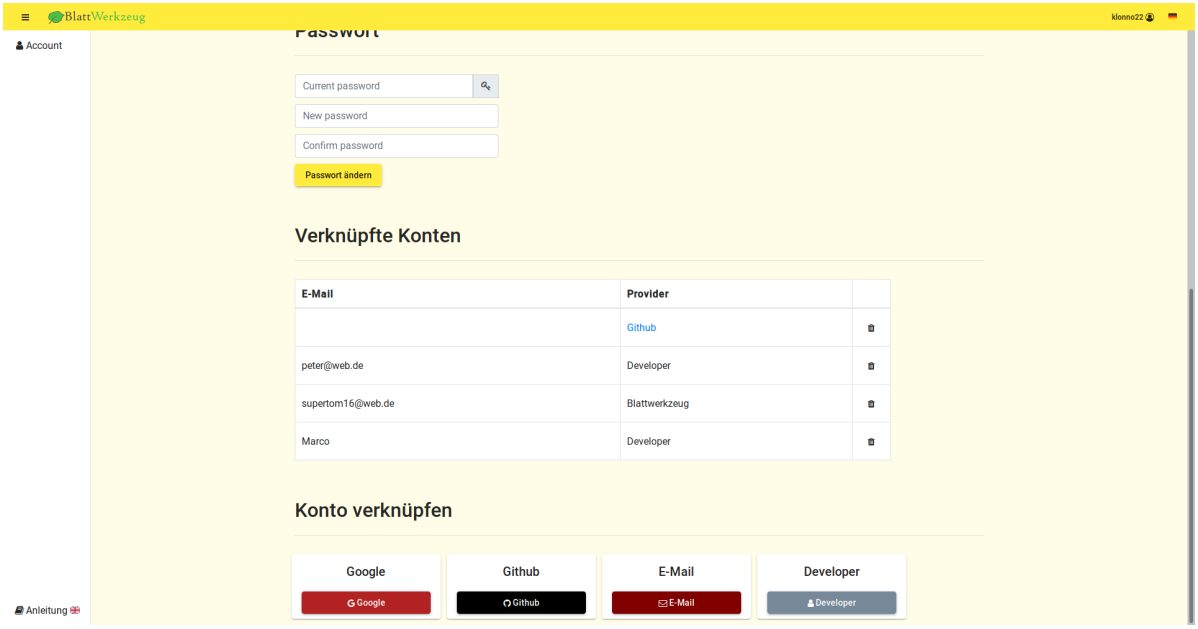


Abbildung 16: Standart und erweiterte Ausführung eines validate-input

der Angular internen *If Direktive zu lösen. Eine weitere Möglichkeit wäre, die zu den Rollen angepassten Codeabschnitte jeweils in eigene Komponenten auszulagern. Ebenso ist es Möglich eine Wrapper-Komponente zu entwickeln, die dynamisch den Inhalt ihres Templates lädt und hierbei bedingt zwischen dem übermittelten Inhalt unterscheiden kann.

Das Nutzen der *If Direktive Angulars für jegliche Codeabschnitte, hat den Nachteil, dass erhöhte Coderedundanz vorkommt. Jede Komponente die Codeabschnitte abhängig von einer Rolle besitzt, müsste auf die jeweiligen Rollen eines Benuzers Zugriff haben. Dafür müsste in jede Komponente ein Service mit eingebunden werden, der die derzeitigen Rollen eines Benutzers an die Komponente ausliefert.

Die Unterteilung der einzelnen Codeabschnitte in eigene Komponenten hat den Nachteil das diese nicht flexibel sind. Eigene Komponenten würden sich stark auf die benötigte Rolle fokusieren. Sollte der Name dieser Rolle geändert werden, müsste nicht nur die Rollen-Abfrage verändert werden, sondern zusätzlich der jeweilige Name der Komponente.

Eine Wrapper-Komponente die dynamisch ihren Inhalt lädt und gleichzeitig bedingt auf den Inhalt achtet, verhindert erhöhte Coderedundanz und ermöglicht eine einfache

Erweiterbarkeit oder Veränderung der Komponente. Aus diesen Gründen wurde sich für diese Methode entschieden.

may_perform

Bedienelemente, die von der jeweiligen Benutzerrolle abhängen, wie beispielsweise das Bedienelement zum Speichern eines Projektes, werden mittels may_perform Wrapper-Komponente überprüft und dargestellt. Da es sich bei Projekten um Resourcen handelt, die nur bei spezifischer Berechtigung bearbeitet werden dürfen, müssen die Bedienelemente der jeweiligen Resource ausgeblendet werden. Sollten die Bedienelemente weiterhin dargestellt werden, kann dies zu einer schlechten User Experience, da ein Benutzer erst nach der Benutzung des Bedienelementes über eine unzureichende Berechtigung informiert wird. Damit eine Überprüfung serverseitig stattfinden kann, wurde sich vorerst mit den zu übermittelnden Daten beschäftigt. Für die Überprüfung der jeweiligen Funktion eines Bedienelementes, muss der Client die auf dem Server auszuführende Funktion angeben. Hierbei musste darauf geachtet werden, dass serverseitig nur Funktionen einer Policy ausgeführt werden können (Sektion 4.1.6). Desweiteren benötigt der Server die Informationen auf welcher Ressource diese Funktion ausgeführt werden soll. Sollte es sich bei der Überprüfung um eine spezifische Ressource handeln, muss ebenfalls die ID der Ressource übermittelt werden.

```
1 export interface MayPerformRequestDescription {
2     resourceType: string;
3     resourceId?: string;
4     policyAction: string;
5 }
```

Listing 14: Interface der zu übermittelnden Daten

Die may-perform Daten wurden in eine Basisklasse und verschiedene abgeleitete Klassen unterteilt. Dabei beinhaltet die Basisklasse jegliche Daten, die Resourcenübergreifend sind. Die abgeleiteten Klassen beinhalten spezialisierte Daten für eine Resource (Abbildung 17). Der Service, der zum Aufruf dieser Daten injiziert wird, erstellt beim Aufruf einer Funktion, eine Instanz dieser Klasse. Ein Vorteil dieses Musters ist, dass die Struktur der Daten eindeutig ist, da die abgeleiteten Klassen jeweils einer Ressource entsprechen. Die Funktionen innerhalb des Services werden nach den jeweiligen Resourcen benannt. Daraus resultiert ein eindeutiges Aufruf-Muster, welches den Ressourcen-Namen gefolgt von der Aktion enhält (Listing

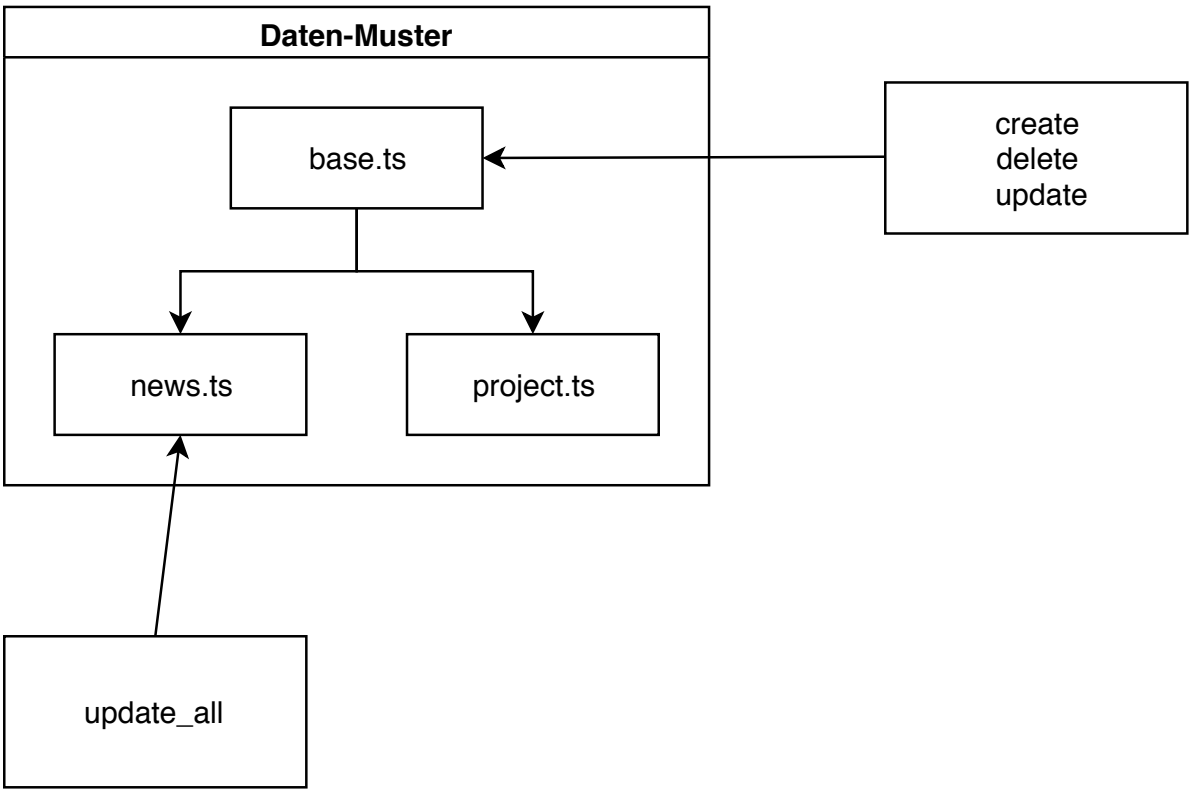


Abbildung 17: Struktur des Daten-Musters

15).

```
1  import { Component } from '@angular/core';
2
3  import { PerformDataService } from '../shared/authorisation/perform-data.service';
4  @Component({
5      templateUrl: "./templates/example.html"
6  })
7  class ExampleComponent {
8      constructor(
9          private _performData: PerformDataService
10     ) { }
11
12     readonly _newsId = "fda587fa-ba7f-11e9-a2a3-2a2ae2dbcce4";
13     readonly performUpdateData = this._performData.news.update(this._newsId);
14     readonly performCreateData = this._performData.news.create();
15 }
```

Listing 15: Beispiel Aufruf zum erhalten der News-Daten

Schlussendlich wird ein Boolscher-Wert an den Client ausgeliefert. Sollte der Boolsche-Wert true zurückliefern, wird das Bedienelement dargestellt.

is-logged-in

Ein Besucher erhält bei einem Aufruf der Blattwerkzeug-Seite Informationen über seinen derzeitigen Benutzer. Die übermittelten Informationen enthalten die Benuzter-Id, die Rollen, die primäre E-Mail und den Benutzernamen. Aus diesem Grund benötigen einige Bedienelemente keine serverseitige Berechtigungs-Abfrage. Das Nutzen der may-perform Komponente wäre in diesem Fall eine überflüssige MaSSname, da diese Bedienelemente ausschlieSSlich auf eine statische Rolle geprüft werden.

Die is-logged-in Wrapper-Komponente nutzt die vom Server übermittelten Rollen zur Überprüfung eines angemeldeten Benutzers. Ein angemeldeter Benutzer, wird clientseitig an seiner Rolle identifiziert. Hierfür werden die übermittelten Rollen auf eine guest Rolle geprüft. Das Vorhandensein einer guest Rolle, stellt einen unangemeldeten Benutzer dar. Die Darstellung eines umschlossenen HTML Codeabschnittes mittels is-logged-in hängt von dem derzeitigen Anmeldestatus eines Benutzers ab.

4.2.6 Routing-Guards

Derzeit ist das Navigieren auf jede Blattwerkzeug-Seite möglich. Routing Guards in Angular bieten die Möglichkeit das Navigieren einzuschränken. Hierfür stellt Angular verschiedene Typen an Routing Guards zur Verfügung.

CanActivate

Das CanActivate Interface von Angular wird genutzt um eine Bedingte Navigierung zu ermöglichen. Hierbei steht das CanActivate Interface für eine Überprüfung einer aktivierten Route. Eine Route gilt dann als aktiviert, wenn der Routing Guard mit dem CanActivate Interface einen Wahrheitswert zurückliefert. Der Fehlschlag eines CanActivate Routing Guards verhindert jedoch nicht das Laden von Routen gebudenen Modulen.

CanLoad

Das CanLoad Interface von Angular wird für das Bedingte Laden von Modulen verwendet. Im Gegensatz zum CanActivate Interface werden Routen gebudene Module, beim Fehlschlag eines CanLoad basierenden Routing Guards, nicht geladen.

Für die Einschränkung der Navigation auf Blattwerkzeug, wurden verschiedene Routing Guards erstellt.

AdminGuard

Damit Bereiche wie beispielsweise der Adminbereich ausschlieSSlich von Administratoren besucht werden können, wurde ein AdminGuard implementiert. Für die Überprüfung des AdminGuards werden die statischen Rollen eines Benutzers auf eine admin Rolle geprüft. Sollte dem derzeitigen Benutzer keine admin Rolle zugewiesen sein, wird dem Benutzer eine Fehlermeldung angezeigt. Die Fehlermeldung wurde mittels Asynchroner-Funktion und einem Dialog-Fenster implementiert. Das Dialog-Fenster bietet die Möglichkeit, mittels afterClosed Funktion, ein Observable als Rückgabewert festzulegen.

Zitat:

> "Ein Observable ist die Repräsentation einer beliebigen Menge von Werten, die über eine beliebige Zeitdauer verteilt sein könnnen."[7]

Das SchlieSSen des Dialog-Fensters löst das zurückgegebene Observable aus. Sobald das Observable ausgelöst wurde, wird auf die Startseite zurück navigiert.

LoggedInGuard

Der LoggedInGuard ähnelt in seiner Funktionsweise dem AdminGuard. Allerdings wird dieser Routing Guard zur Überprüfung eines eingeloggten Benutzers verwendet. Die Einstellungen für Sicherheit und Login dürfen beispielsweise ausschlieSSlich als angemeldeter Benutzer eingesehen werden. Sollte ein unangemeldeter Benutzer eine von einem LoggedInGuard geschützte Route besuchen, erhält dieser die Möglichkeit sich vorerst anzumelden. Da der LoggedInGuard sich ausschlieSSlich im Dialog-Fenster und der zu überprüfenden Rolle von dem AdminGuard unterscheidet, wird die Implementierung nicht weiter erläutert.

MasterGuard

Bevor der AdminGuard oder der LoggedInGuard fehlschlägt, werden jeweils Dialog-Fenster geöffnet. Sollten mehrere Routing Guards für eine Route festgelegt werden, werden beim fehlschlagen der Routing Guards die Dialog-Fenster übereinander gelegt. Die Ursache dafür ist das, von Angular festgelegte, gleichzeitige Ausführen

[7] `https://www.buschmais.de/wp-content/uploads/2019/01/01-2019-Java-aktuell_AUTOR_Michael-Ruttka_Einfuehrung-in-RxJS.pdf`

von Routing Guards. Aus diesem Grund muss ein Algorithmus zum ausführen nacheinander gefolgter Routing Guards implementiert werden.

Der MasterGuard implementiert einen Algorithmus zum ausführen nacheinander gefolgter Routing Guards. Hierfür werden die auszuführenden Routing Guards im data Objekt des Routes Typen hinterlegt. Sobald der MasterGuard aufgerufen wird, wird über die von Angular übergebene Route auf das data Objekt zugegriffen. AnschlieSSend wird von jedem, im data Objekt enhaltenen, Routing Guard eine Instanz erstellt und die canActivate Methode aufgerufen. Sollte ein Routing Guard felschlagen, schlägt der Master Guard ebenfalls fehl.

5 Fazit

Das Fazit befasst sich kurz mit den erreichten Zielen und den möglichen Erweiterungen die aus dieser Thesis resultieren.

5.1 Erreichte Ziele

Im Rückblick auf die Anforderungen dieser Arbeit wurden alle Ziele erreicht. Dabei ist eine Anmeldung mittels Passwort und oAuth2 realisiert worden. Die Anmeldung mittels oAuth2 ist über die Provider Google und GitHub möglich. Für eine Anmeldung mittels Passwort wurde eine zusätzliche Möglichkeit zur Registrierung hinzugefügt.

Ebenfalls realisiert werden konnte die serverseitige Überprüfung von Benutzerrechten. Für das Erstellen, Löschen und Modifizieren von Projekten oder News wurde diese Überprüfung bereits implementiert. Implementiert wurde auch der bedingte Zugriff auf clientseitige Routen und Bedienelemente.

Ebenso implementiert werden konnten die Einstellungen im Bereich Sicherheit und Login. Mittlerweile ist es Möglich sein bereits erstelltes Konto mit weiteren Konten zu verknüpfen. Zusätzlich können die bereits verknüpften Konten verwaltet werden. Der Wechsel eines Passwortes und eines Benutzersnamens ist inzwischen ebenfalls durchführbar.

5.2 Ausblick

Im Hinblick auf die Zukunft haben sich während der Ausarbeitung der Thesis bereits mehrere Ideen zur Weiterentwicklung ergeben.

Zu diesem Zeitpunkt ist die Autorisierung von Bedienelementen innifizient, weSShalb für die Zukunft bereits eine Weiterentwicklung der zuversendenen HTTP-Anfragen geplant ist. Hierbei werden HTTP-Anfragen innerhalb von einer Sekunde zusammen gefasst und als eine Anfrage versendet. Folglich würde die Serverauslastung, durch übermäSSige Anfragen, reduziert werden.

Desweiteren ist das Hinzufügen weiterer statischer Rollen geplant, die spezifisch für beispielsweise Lehrer oder Schüler angewendet werden. Da Blattwerkzeug weitestgehend an Schulen auftreten soll, kann mit der Möglichkeit zur Autorisierung weiterer Inhalt für unterschiedliche Benutzergruppen erstellt werden.

Eine weitere geplante Weiterentwicklung ist das Anlegen einer Profilseite für jeden registrierten Benutzer. Die Einstellungsmöglichkeiten eines Profils könnten hierbei in dem bereits erstellten Einstellungs-Modul ergänzt werden.

Darüber hinaus bietet die Implementierung von Authentifizierung und Autorisierung weitaus mehr Möglichkeiten weitere Entwicklungen an Blattwerkzeug vorzunehmen. Ein Beispiel für eine derzeit nicht geplante Erweiterung wäre ein Nachrichten-System zur Kommunikation zwischen Benutzern.