Entwicklung einer Applikation zur Lizenzprüfung

Schweizerischer Judo und Ju-Jitsu Verband (SJV)

BACHELORARBEIT

CYRIL MICHEL

September 2017

Betreuung

Prof. Dr. Jacques PASQUIER–ROCHA
UND
PASCAL GREMAUD

Software Engineering Group

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Software Engineering Group
Department of Informatics
University of Fribourg
(Switzerland)

Kurzzusammenfassung

Damit der Judo- und Jiu-Jitsu-Verband bei Kursen eine Lizenzprüfung durchführen kann, wird eine technische Lösung benötigt.

In dieser Arbeit wird diese Lösung erarbeitet und umgesetzt. Dabei steht eine Client-Server-Architektur mit einer REST-Schnittstelle im Vordergrund.

Im Rahmen dieses Kleinprojekts werden in einem ersten Schritt die Kundenanforderungen erhoben und analysiert und im weiteren Verlauf technisch umgesetzt. Dazu wird für die Kommunikation zwischen Backend und Frontend eine REST-Schnittstelle verwendet.

Zur Datenspeicherung wird eine NoSQL-Datenbank verwendet, welche in einem kleinen theoretischen Abschnitt thematisiert wird.

Schlüsselworte: Angular 2, Express, Node.js, Service Web, REST, Web Token, MongoDB, Client-Server-Architektur, Webanwendung

Präambel

Vorwort

Das vorliegende Dokument ist Bestandteil der Bachelorarbeit von Cyril Michel. Diese wurde im Rahmen des Studiums der Wirtschaftsinformatik an der Universität Freiburg erstellt.

Danksage

Dank Professor Pasquier konnte ich eine Arbeit erstellen, welche für mich spannend und lehrreich wahr. Ich möchte mich bei ihm für seine Unterstützung und Betreuung bedanken.

Des Weiteren möchte ich mich auch bei Pascal Gremaud bedanken. Ich konnte ihm jederzeit Fragen stellen und erhielt hilfreiche Antworten und Lösungsvorschläge. Dank seiner Unterstützung und Kompetenz konnte ich diese Arbeit abschliessen.

Abschliessend gilt ein grosser Dank der operativen Leitung des SJV, welche mein Vorhaben unterstützt hat.

Notationen und Konventionen

Inhaltsverzeichnis, Abbildungsverzeichnis und Tabellenverzeichnis befinden sich am Anfang des Dokuments und verweisen auf alle entsprechenden Elemente.

Im Anhang befindet sich ein zusätzlicher Codeausschnitt, welcher aus Platzgründen nicht im Text eingebunden ist.

Am Ende des Dokumentes befinden sich die Quellenangaben, auf welche im Text verwiesen wird.

Datenschutz

Die Personendaten in der Datenbank stehen unter Datenschutz. Sie dürfen weder verbreitet, noch auf sonstige Art dupliziert oder verwendet werden.

Inhaltsverzeichnis

1 Einleitung **1**

2 Anforderungen und Planung **2**

2.1 Projektplan ... 2

2.2 Benutzeranforderungen ... 4

2.2.1 Use Cases ... 5

2.2.2 Technische Anforderungen ... 7

2.3 Qualitätssicherung ... 8

2.3.1 Testplanung ... 8

2.3.2 Testergebnisse ... 8

Aufgetretene Fehler ... 8

Schlussresultate ... 9

2.4 Systemarchitektur ... 10

2.4.1 Client-Server-Architektur mit REST-Schnittstelle ... 11

2.4.2 Server ... 12

2.4.3 Datenbank ... 13

2.4.4 Backend ... 13

2.4.5 Frontend ... 13

3 Applikationsentwicklung **14**

3.1 Kenntnisaneignung ... 14

3.2 Entwicklungswerkzeuge ... 14

3.3 Ordnerstruktur ... 15

3.4 Entwicklung des Backends ... 16

3.4.1 REST-Schnittstellen ... 16

3.4.2 Interessante Aspekte im Code ... 19

Lizenzierte suchen ... 19

Registrierung ... 21

Authentifizierung ... 22

Person löschen ... 23

3.4.3 Testing mit Postman ... 24

3.5 Entwicklung des Frontends ... 25

3.5.1 Interessante Aspekte im Code ... 25

Person registrieren ... 26

Personensuche 29
Personen löschen 30
Abmelden / Logout 30
4 Endprodukt **32**
4.1 Benutzeroberfläche 32
4.2 Anwendungs-Lifecycle 36
4.2.1 Anwendungsinitialisierung 36
4.2.2 Iterative Phase der Jahresnutzung 37
4.3 Zielerreichung 37
4.4 Weiteres Vorgehen 38
5 Exkurs NoSQL **39**
5.1 Einleitung 39
5.2 Geschichte und Begrifflichkeit 39
5.3 MongoDB und Anwendung im Projekt 41
6 Rückblick **42**
A Source Code **1**
B Quellen **4**
C Web Quellen **5**
D Verwendete Tutorials **6**
E Verwendetes Template **7**

Abbildungsverzeichnis

Abbildung 1: Projektplan 3
Abbildung 2: Use-Case-Diagramm 5
Abbildung 3: Systemarchitektur 11
Abbildung 4: Server-Spezifikationen bei AWS 12
Abbildung 5 Ordnerstruktur 15
Abbildung 6: Dateistruktur Frontend 16
Abbildung 7: Kommandozeilenbefehl ng new 16
Abbildung 8: Klassendiagramm Backend 19
Abbildung 9: Ausschnitt aus users.js 20
Abbildung 10: Ausschnitt aus user.js 20
Abbildung 11: Ausschnitt aus users.js 21
Abbildung 12: Ausschnitt aus user.js 22
Abbildung 13: Ausschnitt aus users.js 22
Abbildung 14: Ausschnitt aus users.js 23
Abbildung 15: Ausschnitt aus users.js 23
Abbildung 16: Ausschnitt aus user.js 24
Abbildung 17: Ansicht Postman Headers 24
Abbildung 18: Ansicht Postman Body 25
Abbildung 19: Kommandozeilenbefehl ng g componet 25
Abbildung 20: Kommandozeilenbefehl ng g service 26
Abbildung 21: Ausschnitt aus validate.service.ts 26
Abbildung 22: Ausschnitt aus validate.service.ts 27
Abbildung 23: Ausschnitt aus validate.service.ts 27
Abbildung 24: Ausschnitt aus validate.service.ts 27
Abbildung 25: Ausschnitt aus validate.service.ts 28
Abbildung 26: Ausschnitt aus auth.service.ts 28
Abbildung 27: Ausschnitt aus personen-suche.component.ts 29
Abbildung 28: Ausschnitt aus personen-suche.component.html 29
Abbildung 29: Ausschnitt aus personen-suche.component.html 30

Abbildung 30: Ausschnitt aus navbar.component.ts 30
Abbildung 31: Ausschnitt aus auth.service.ts 30
Abbildung 32: Login-Ansicht 32
Abbildung 33: Ansicht Personensuche 33
Abbildung 34: Ansicht Personensuche Admin 34
Abbildung 35: Ansicht zur Personenerfassung 35
Abbildung 36: Anwendungs-Lifecycle 36

Tabellenverzeichnis

Tabelle 1: Rollenkonzept 5
Tabelle 2: Beschreibung UC1 6
Tabelle 3: Beschreibung UC2 6
Tabelle 4: Beschreibung UC3 7
Tabelle 5: Beschreibung UC4 7
Tabelle 6: Beschreibung UC5 7
Tabelle 7: Port-Zuweisung 12
Tabelle 8: Schnittstelle «Personenregistrierung» 17
Tabelle 9: Schnittstelle «Personenregistrierung zwei» 17
Tabelle 10: Schnittstelle «Login» 18
Tabelle 11: Schnittstelle «Personensuche» 18
Tabelle 12: Schnittstelle «Person löschen» 18
Tabelle 13: Schnittstelle «Profil anzeigen» 18

1 Einleitung

Der Schweizerische Judo- und Ju-Jitsu-Verband führt jährlich zirka 350 Kurse für seine lizenzierten Mitglieder durch. Dabei ist das Besitzen einer Jahreslizenz eine Bedingung, um an diesen Kursen teilnehmen zu dürfen.

In der Praxis wurde festgestellt, dass das Überprüfen dieser Lizenzen relativ aufwendig ist, da die Lizenzkarten von den Teilnehmern selten mitgeführt werden.

Der SJV verfügt über eine eigene Fachanwendung, in welcher die Lizenzen verwaltet werden. Diese steht den Kursverantwortlichen jedoch nicht zur Verfügung. Das Erweitern dieser Fachanwendung um diese Funktion ist für den SJV keine Option.

Im Rahmen dieser Arbeit wird eine IT-Lösung für diese Problemstellung erarbeitet.

Die angedachte Lösung bietet eine Möglichkeit, die Lizenzen dezentral zu kontrollieren. Eine Suche in den replizierten Daten mit Namen und Geburtsdatum erlaubt es, herauszufinden, ob eine Person lizenziert ist oder nicht.

2 Anforderungen und Planung

2.1 Projektplan

Die Identifikation aller für das Projekt notwendiger Aktivitäten, sowie des zeitlichen Bedarfes dafür, prägten den Beginn der Arbeit.

Um einen optimalen Ablauf des Projektes zu gewährleisten, war ein Projektplan unabdingbar. Mit Hilfe dieses Planes konnte die Zeit von Anfang an sinnvoll eingeteilt werden und es wurde ersichtlich, welche Aufgaben bis zur finalen Abgabe zu erfüllen waren. Ein weiterer Vorteil der Projektplanung mit Hilfe eines Gantt-Diagrammes ist, dass ersichtlich wird, welche Arbeit von welchem vorhergehenden Schritt abhängig ist. Dies ermöglicht, die Prioritäten passender zu setzen. Falls bei einem Punkt es zu Problemen kommt, findet man schnell eine andere Aufgabe, die parallel erledigt werden kann.

Der nachstehende Projektplan in Abbildung 1 ist die finale Version, welche während des Projektes fortlaufend angepasst wurde, um auf allfälligen Verschiebungen oder Änderungen reagieren zu können.

Da es sich um ein kleines Projekt handelt, in dem die Benutzeranforderungen von Anfang an relativ klar waren, wurde der Projektplan dem Wasserfallmodell aufgestellt. Dies ist angelehnt an die Projektmanagement-Methode Hermes 5. Die Phaseneinteilung sowie die für das Projekt sinnvollen Ergebnisse wurden dieser Methode entnommen, ohne sich strikt an alle Aspekte der Methode zu halten.

Im Projektplan ist bei den Tagesangaben zu beachten, dass die angegebene Zeit die maximal zur Verfügung stehende Durchlaufzeit der Aufgabe repräsentiert. Es wurde also nicht während der ganzen Zeit mit voller Kapazitäten an dieser Arbeit gearbeitet; somit wurden in manchen Phasen relativ viele Tage für kleinere Arbeiten aufgewendet, und umgekehrt.

Vorgangsname	Dauer	Anfang	Ende	Vorgänger
INITIALISIERUNG	**44 Tage**	**01.12.16**	**31.01.17**	
Projektidee erarbeiten, besprechen	15 Tage	01.12.16	21.12.16	
Machbarkeit prüfen	15 Tage	22.12.16	11.01.17	3
Projektplanung	10 Tage	12.01.17	25.01.17	4
Freigabe der Bachelorarbeit	1 Tag	31.01.17	31.01.17	5
KONZEPT	**20 Tage**	**06.02.17**	**03.03.17**	**2**
Erheben Benutzeranforderungen	5 Tage	06.02.17	10.02.17	
Definition Use Case	5 Tage	13.02.17	17.02.17	8
Erstellen Systemarchitektur	10 Tage	20.02.17	03.03.17	9
Definition Entwicklungswerkzeuge	10 Tage	20.02.17	03.03.17	9
REALISIERUNG	**139 Tage**	**06.03.17**	**14.09.17**	**7**
Anneignung Entwicklungsmethoden	98 Tage	06.03.17	19.07.17	
Installation / Konfiguration Entwicklungsumgebung	10 Tage	17.04.17	28.04.17	
Installation / Konfiguration Datenbank	10 Tage	17.04.17	28.04.17	
Entwicklung Backend	20 Tage	01.05.17	26.05.17	15;14
Modul Test	20 Tage	01.05.17	26.05.17	16AA
Entwicklung Frontend	20 Tage	29.05.17	23.06.17	16
Modul Test	20 Tage	29.05.17	23.06.17	18AA
Systemtest	10 Tage	26.06.17	07.07.17	16;18
Funktionstest	10 Tage	10.07.17	21.07.17	20
SW Korrektur	20 Tage	26.06.17	21.07.17	20AA
SW Vorabnahme	1 Tag	27.07.17	27.07.17	20;21;22
Dokumentation Bachelorarbeit	25 Tage	17.07.17	18.08.17	
Finalisierung Lieferobjekte	10 Tage	28.08.17	08.09.17	24EA+5 Tage
Abgabe Termin	1 Tag	14.09.17	14.09.17	25
Einführung	**1 Tag**	**02.10.17**	**02.10.17**	**12**
Out of scope Bachelor Arbeit	1 Tag	02.10.17	02.10.17	

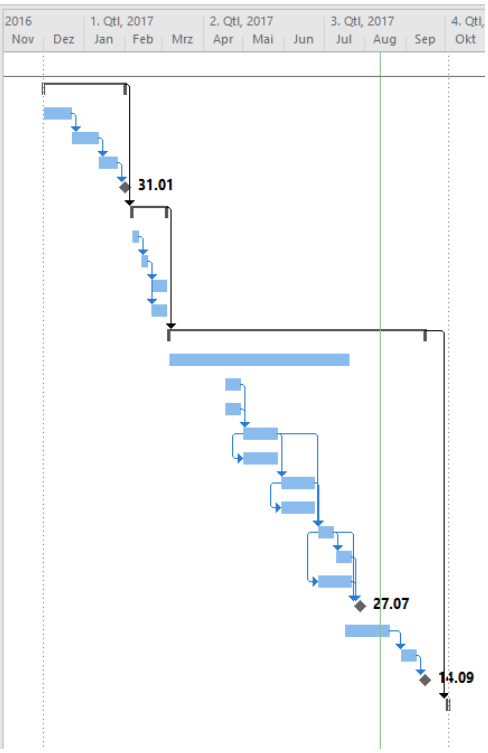


Abbildung 1: Projektplan

Initialisierung

Bei der Initialisierung handelt es sich um die richtungsentscheidende Phase des Projekts. Es wurde definiert, was die Bachelorarbeit beinhalten muss und in welche Richtung es gehen sollte. Dabei ist auch wichtig, zu definieren, welche Arbeiten nicht oder eventuell erst zu einem späteren Zeitpunkt erledigt werden. Dies ist relevant, um die Technologiewahl so planen zu können, dass die Applikation später erweitert werden kann.

Konzept

In der Konzeptphase geht es darum, detaillierter zu bestimmen, was genau gefragt ist und mit welchen Werkzeugen dies erreicht werden kann. Ein zentraler Bestandteil der Konzeptphase in dieser Arbeit war das Errichten der Systemarchitektur, denn die gesamte Implementierung basiert auf dem Zusammenspiel der verschiedenen Komponenten.

Die Entscheidungen, welche Technologien verwendet werden sollten, wurden grösstenteils in dieser Phase getroffen. Bei der Aneignung von Kenntnissen zu unterschiedlichen Technologien wurden diese Entscheidungen entweder gefestigt oder gegebenenfalls überdacht und angepasst, falls die Anforderungen anders besser abgedeckt werden konnten.

Realisierung

Wie im Projektplan ersichtlich, bildet die Realisierung den Kernteil des Projektes. Sie beinhaltet die Entwicklung sowohl des Frontends wie auch des Backends. Dazu gehört auch das Testen und das Erstellen der Dokumentation.

Einführung

Jede neue Softwarelösung braucht eine Einführung. Dies gilt auch für die hier vorgestellte Applikation. Diese Einführung und der dazugehörige *Early Life Support* sind jedoch nicht Bestandteil dieser Bachelorarbeit und werden nicht in diesem Dokument behandelt.

2.2 Benutzeranforderungen

Wie in der Einleitung angesprochen wurde, ist das Ziel dieser Arbeit, eine Applikation für den SJV zu erstellen, damit Lizenzen bei der Durchführung von Kursen kontrolliert werden können.

Damit bei der Entwicklung die Bedürfnisse des Sportverbandes abgedeckt werden, muss zunächst eine Bedürfnisanalyse erfolgen.

Um diese Bedürfnisse zu erfassen, wurde ein Gespräch mit den zuständigen Personen beim SJV geführt. In diesen Diskussionen ergaben sich die unterschiedlichen Anforderungen an das System.

Es sollte eine Applikation geben, mit deren Hilfe alle Kursveranstalter überprüfen können, ob jemand eine zum gegebenen Zeitpunkt gültige Lizenz besitzt. Wichtig dabei war den zuständigen Personen beim SJV, dass nicht jeder Zugriff auf diese Daten haben sollte. Vielmehr sollte zur Identifikation einer Person zusätzlich zu Name und Vorname das Geburtsdatum angegeben werden. Die Anwendung sollte auf mobilen Endgeräten lauffähig sein, damit sie mit möglichst geringem Aufwand überall verwendet werden kann.

Dies sind die funktionalen Anforderungen an die Anwendung aus Sicht der Endnutzer.

Zusätzlich zu diesen Anforderungen gab es Anforderungen für die Wartung und Pflege der Daten, welche durch die SJV-Administration zentral erfolgt. So muss es für SJV-Mitarbeitende die Möglichkeit geben, Personen aus der Datenbank zu löschen und zu ergänzen. Diese Anforderungen entstehen daraus, dass es einerseits möglich ist, dass eine Person sich erst im Laufe des Jahres eine neue Lizenz erstellt; anderseits kann es sein, dass aus disziplinarischen Gründen eine Lizenz entzogen wird.

Das Ergänzen und Entfernen von Einzelpersonen sollte durch einen SJV-Mitarbeiter ohne spezifische IT-Kenntnisse durchgeführt werden können.

Zusätzlich zu dieser Anforderung muss es möglich sein, einmal im Jahr einen vollständigen Datenstand der vorhandenen Lizenzen zu importieren. Im Vorfeld muss dazu der vorhandene Datenstand gelöscht werden. Dies wird durch den IT-Verantwortlichen des SJV ausgeführt.

Aus den oben beschriebenen Anforderungen kann ein einfaches Berechtigungs- und Rollenkonzept abgeleitet werden, welches in Tabelle 1 dargestellt ist.

Rolle	Berechtigung
Kursveranstalter	Daten suchen
Administratoren	Daten suchen Daten einfügen Daten löschen

IT-Verantwortliche des SJV	Daten suchen Daten einfügen Daten löschen Jahresimport durchführen

Tabelle 1: Rollenkonzept

Während der Diskussion mit den SJV-Mitarbeitern wurde relativ schnell deutlich, dass die ganze Applikation so intuitiv wie möglich sein sollte, denn bei den Kursveranstaltern handelt es sich meist um Personen, welche im Umgang mit digitalen/IT-Systemen möglicherweise unerfahren sind.

Beim Ausarbeiten dieser Vorgaben ergaben sich zwei Arten von Anforderungen: die funktionalen und die nichtfunktionalen Anforderungen. In den nächsten zwei Kapiteln werden zuerst die funktionalen Anforderungen in Use Cases beschrieben. Anschliessend werden im Kapitel 2.2.2 «Technische Anforderungen» die nichtfunktionalen Anforderungen erörtert.

2.2.1 Use Cases

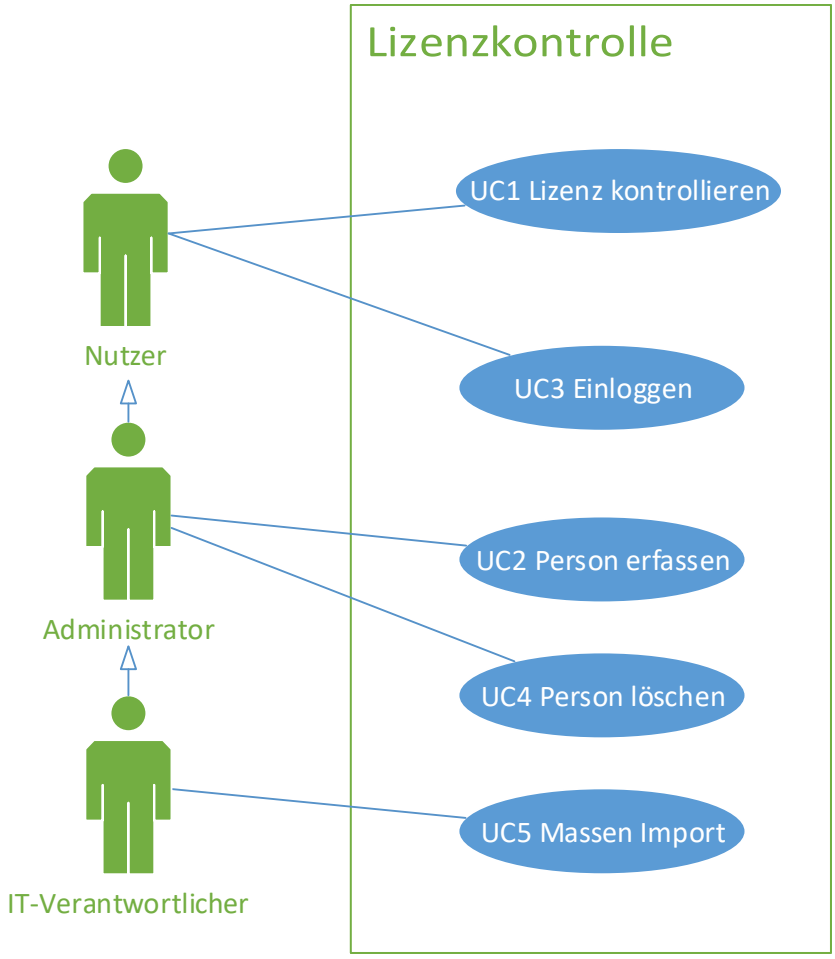


Abbildung 2: Use-Case-Diagramm

In den nachstehenden Tabellen 2 bis 6 werden die Anwendungsfälle, welche im Use-Case-Diagramm (Abbildung 2) dargestellt wurden, detailliert beschrieben.

Name	**Lizenz kontrollieren**
Id	UC1
Beschreibung	Es muss möglich sein, eine Person über ihren Namen und Vornamen zu suchen und mittels Geburtsdatum eindeutig zu identifizieren. Bei der Person muss ersichtlich sein, ob sie eine aktuell gültige Jahreslizenz besitzt.
Input	Name und / oder Vorname
Output	Angabe, ob die Person lizenziert ist
Akteure	Jede Person mit gültigen Logindaten muss Zugang haben

Tabelle 2: Beschreibung UC1

Name	**Person erfassen**
Id	UC2
Beschreibung	Einzelne Personen müssen direkt über dem Webzugang erfasst werden können.
Input	Personendaten: - Name (Pflichtfeld) - Vorname (Pflichtfeld) - Geburtstag - SJV Nr. - E-Mail-Adresse - Rolle (Pflichtfeld) - Lizenziert (ja / nein) (Pflichtfeld) - Passwort
Output	Bestätigung, dass die Person erfasst und gespeichert wurde
Akteure	Administrator

Tabelle 3: Beschreibung UC2

Name	**Einloggen**
Id	UC3
Beschreibung	Damit nur berechtigte Personen die Daten entsprechend ihrer Rolle sehen oder bearbeiten können, müssen sie sich einloggen können.
Input	E-Mail-Adresse, Passwort

Output	Person wird eingeloggt und dies wird ihr bestätigt
Akteure	Alle Personen mit Zugangsrechten

Tabelle 4: Beschreibung UC3

Name	**Person löschen**
Id	UC4
Beschreibung	Einzelne Personen müssen direkt über den Webzugang gelöscht werden können.
Input	Löschanfrage
Output	Bestätigung, dass die Person gelöscht wurde
Akteure	Administrator

Tabelle 5: Beschreibung UC4

Name	**Massen Import**
Id	UC5
Beschreibung	Einmal im Jahr müssen alle alten Daten gesammelt gelöscht werden und die neuen Daten importiert werden. Dabei handelt es sich um alle lizenzierten Personen des SJV für eine Saison.
Input	Alle Personendaten
Output	Alle Personen gespeichert
Akteure	IT-Verantwortlicher des SJV

Tabelle 6: Beschreibung UC5

2.2.2 Technische Anforderungen

Die Ansprechpartner beim SJV nennen vier technische Anforderungen, welche ihnen wichtig sind:

- Mobilefähig – die Anwendung sollte auf mobilen Endgeräten funktionieren
- Keine langen Ladezeiten
- Zugangsgeschützt
- System ist Fehler unanfällig

Zusätzlich zu diesen Anforderungen sind auch klassische technische Anforderungen wie Skalierbarkeit und Wartbarkeit essentiell. Für die Skalierbarkeit sind Architekturentscheidungen und die Wahl der Technologie zentral. Auch auf die Wartbarkeit

haben diese Aspekte einen Einfluss. Hier ist jedoch vor allem von Bedeutung, dass der Code und die Dateistrukturen sauber und gut dokumentiert sind.

2.3 Qualitätssicherung

Die Qualitätssicherung, auch Testing genannt, ist ein wichtiger Aspekt einer solchen Arbeit. Es müssen sowohl funktionale als auch nichtfunktionale Kriterien getestet werden. Mit gut geplanten und strukturierten Tests sollte es möglich sein, einen grossen Anteil der Fehler zu finden. Jedoch ist zu bedenken, dass durch Tests keine absolute Sicherheit bezüglich der Fehlerentfernung besteht. Es kann aber garantiert werden, dass die gewünschten Funktionen die definierten Aufgaben erfüllen.

2.3.1 Testplanung

Die Tests werden auf verschiedenen Ebenen zu verschiedenen Zeitpunkte im Projekt durchgeführt. Dies soll gewährleisten, dass mehr Fehler gefunden werden, da die Tests immer in etwas anderen Situationen und mit anderen Sichtwinkeln durchgeführt werden.

- Während der Entwicklung wird jedes Modul so früh wie möglich getestet, um Fehler möglichst früh genau lokalisieren zu können.
- Die REST-Schnittstellen des Backend werden mit dem Werkzeug Postman getestet.
- Sobald das Frontend erstellt ist, wird das Gesamtprojekt durch den Entwickler getestet. Dabei kann nun erstmals auch die Bedienbarkeit getestet werden. Hier ist auch wichtig, dass die Kompatibilität mit verschiedenen Browsern und mobilen Zielplattformen getestet wird.
- Der letzte Test erfolgt durch den Endnutzer. Dabei ist Wert darauf zu legen, dass die Auswahl an Personen möglichst die spätere Nutzergruppe widerspiegelt.

2.3.2 Testergebnisse

Da es sich beim Testen um einen fortlaufenden Prozess handelt, werden an dieser Stelle nicht alle Testresultate dargestellt. In einem ersten Schritt werden einzelne aufgetretene Probleme dargestellt, welche korrigiert werden konnten. In einem zweiten Schritt werden noch die Schlussresultate aufgezeigt. Während der Entwicklungszeit traten viele Fehler auf, bei den hier aufgeführten handelt es sich nur um einen kleinen Ausschnitt. Es werden die spannenden und zentralen Fehler erläutert.

Aufgetretene Fehler

- Es war möglich, den eigenen Benutzer zu löschen. Beim Testen waren noch sehr wenige Administratoren gespeichert. Nach dem das Löschen einige Male getestet wurde, wurde

versehentlich der eingeloggte Administrator gelöscht, wobei es sich um den letzten Administrator handelte.

Da die Passwörter zuerst mittels einer Hash-Funktion hinterlegt waren, konnte nicht einfach ein neuer Benutzer über die MongoDB-Kommandozeile eingefügt werden. Auch über die REST-Schnittstelle war das nicht möglich, da man dafür ein gültiges Web Token benötigte.

Es musste also bei der REST-Schnittstelle die Funktion, welche auf ein Web Token überprüft, entfernt werden. Nachdem die Person hinzugefügt wurde, wurde diese Überprüfung anschliessend wieder eingefügt.

Damit ein Fehlerzustand, der so schwierig zu beheben ist, nicht mehr entstehen kann, wird nun überprüft, dass man sich selber nicht löschen kann.

Die Fehlerursache wurde behoben, indem es unmöglich gemacht wurde, sich selber als Nutzer zu löschen.

- Das Menü wurde auf einem kleinen Bildschirm nicht korrekt angezeigt.

  Durch eine Anpassung des Menüs ist nun auf mobilen Endgeräten die gleiche Darstellung wie bei einem Desktop-Gerät möglich und somit das Problem behoben. Diese Lösung war umsetzbar, da das Menü nur maximal zwei Menüpunkte enthält, die auch auf einem kleinen Display noch gut dargestellt werden können.

- Es war nicht möglich, mittels eines Leerschlags nach Vor- und Nachnamen zu suchen.

  In einer ersten Phase konnte man nur ein Wort eingeben, was bei Namen oft nicht ausreichend ist: Sucht man zum Beispiel nach den Namen «Müller» oder «Florian», erhält man eine relativ lange Liste von Personen.

  Dieses Problem musste behoben werden, sodass die gesuchte Person bei der Eingabe von mit einem Leerschlag getrennten Vor- und Nachnamen gefunden wird. Dabei darf es insbesondere keine Rolle spielen, ob zuerst der Vor- oder der Nachname eingegeben wird.

  Um dies umzusetzen, war eine umfangreiche Korrektur vonnöten, denn der Fehler musste sowohl im Frontend wie auch im Backend korrigiert werden.

Schlussresultate

Nach mehreren Tests mit verschiedenen Personen bleibt noch ein bekannter Fehler offen, der noch nicht behoben ist. Dabei handelt es sich um ein Anzeigeproblem im Internet Explorer. Leider konnte die Ursache des Problems noch nicht gefunden werden. Die Anzeige mit alternativen, aktuellen Versionen von Browsern, wie Firefox und Chrome, funktioniert jedoch.

Ansonsten konnten alle bis dahin bekannten Fehler behoben werden. Nach dem ergänzen von kleinen Hinweismeldungen war die Applikation für die Testpersonen selbsterklärend.

Für die Tests wurden nur aktuelle Versionen der gängigen Browser verwendet – es könnte somit sein, dass bei älteren Versionen nicht alles funktioniert.

2.4 Systemarchitektur

Die Systemarchitektur beschreibt das Zusammenspiel der einzelnen Komponenten der Applikation. Eine Darstellung dieser Architektur ermöglicht einen Überblick und ein tieferes Verständnis dieser Zusammenhäng.

Bei der Erstellung der Architektur wurde ein Top-Down-Ansatz verfolgt: Zuerst wurde definiert, wie und welche Komponenten miteinander kombiniert werden, um die gestellte Aufgabe zu lösen. Erst in einem zweiten Schritt wurden die einzelnen Komponenten genauer definiert. Dies bedeute, dass ebenfalls erst im zweiten Schritt definiert wurde, welche Technologien am besten für die jeweilige Komponente geeignet sind.

Dieser Ansatz sollte ermöglichen, dass die einzelnen Komponenten optimal aufeinander abgestimmt werden konnten, da bekannt war, welche anderen Komponenten noch verwendet werden.

In der nachfolgenden Abbildung 3 ist ein Überblick der Applikation dargestellt. Auf die einzelnen Komponenten wird in einem zweiten Schritt eingegangen.

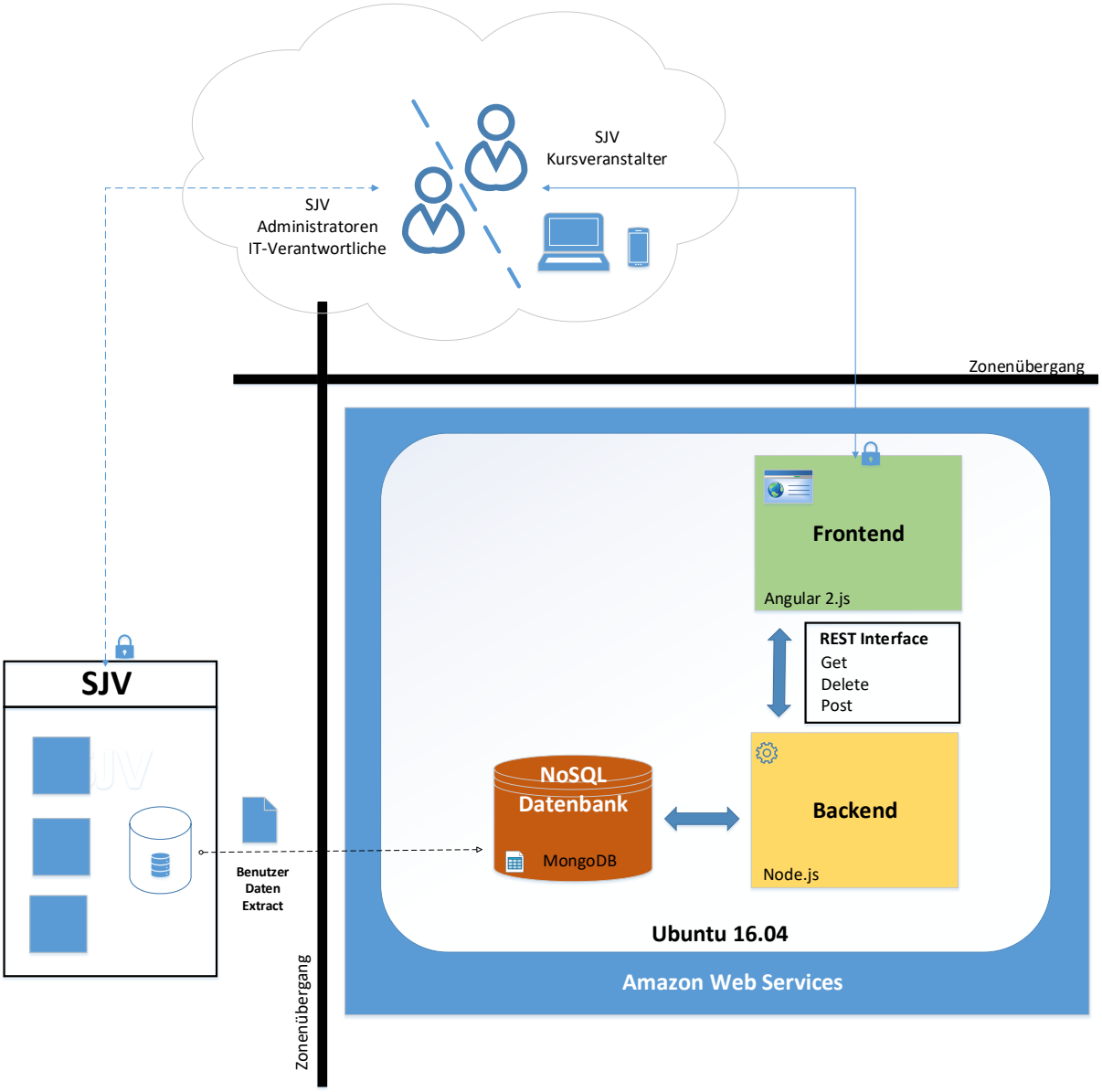


Abbildung 3: Systemarchitektur

2.4.1 Client-Server-Architektur mit REST-Schnittstelle

Von Anfang an war klar, dass eine Client-Server-Architektur errichtet werden sollte. Da in der heutigen Zeit REST-Schnittstellen (Representational State Transfer) weit verbreitet sind, war diese Entscheidung schnell getroffen. Der Vorteil, den eine solche Architektur mit sich bringt, besteht darin, dass der Dienst auch anderen Parteien zur Verfügung gestellt werden kann. Dadurch, dass im World Wide Web der grösste Teil der Anforderungen gegeben ist, ist es für jeden Nutzer einfach zu verwenden.

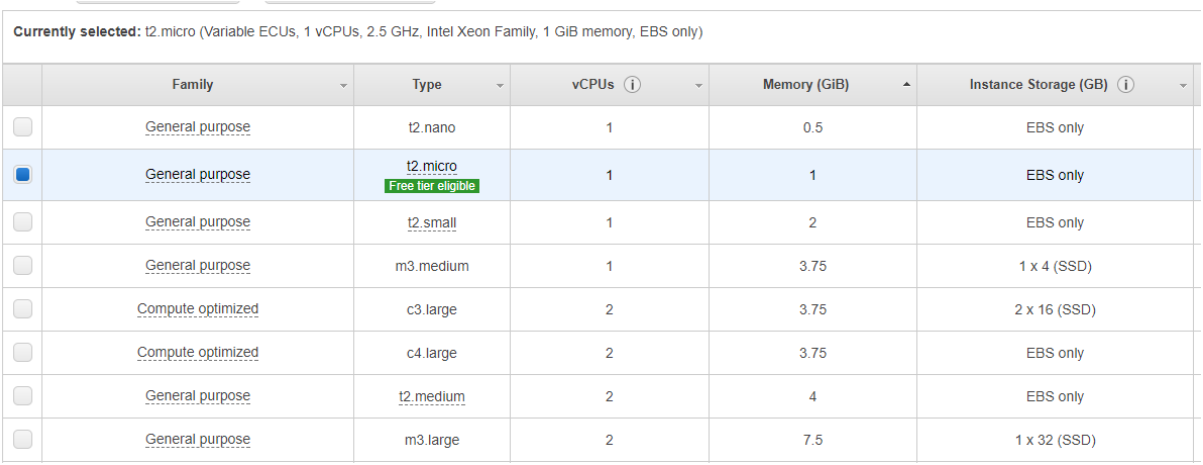
2.4.2 Server

Da der SJV bereits eine Applikation besitzt, welche auf einem virtuellen Server läuft, wurde an dieser Stelle nicht nach anderen Optionen gesucht, sondern derselbe Anbieter von virtuellen Servern gewählt. Dabei handelt es sich um Amazon Web Services (AWS)[Web1].

Bei AWS wurde ein neuer virtueller Server mit dem Betriebssystem Ubuntu 16.04 eingerichtet.

Dabei musste aus verschiedenen möglichen Angeboten ausgewählt werden. In Abbildung 4 sind einige dieser Angebote aufgezeigt. Für diese Anwendung wurde eine Instanz mit einer CPU und 1 GB Arbeitsspeicher gewählt. Diese Ressourcen sollten für den Anwendungsfall ausreichen, denn es werden nie viele gleichzeitige Anfragen erwartet. Somit muss die Rechenleistung nicht besonders hoch sein, auch die Speicherkapazität sollte ausreichen.

Ein grosser Vorteil von AWS besteht darin, dass diese Maschinenkonfigurationen jederzeit angepasst werden können, falls sie den Anforderungen nicht mehr genügen.



Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)
☐	General purpose	t2.nano	1	0.5	EBS only
☑	General purpose	t2.micro Free tier eligible	1	1	EBS only
☐	General purpose	t2.small	1	2	EBS only
☐	General purpose	m3.medium	1	3.75	1 x 4 (SSD)
☐	Compute optimized	c3.large	2	3.75	2 x 16 (SSD)
☐	Compute optimized	c4.large	2	3.75	EBS only
☐	General purpose	t2.medium	2	4	EBS only
☐	General purpose	m3.large	2	7.5	1 x 32 (SSD)

Abbildung 4: Server-Spezifikationen bei AWS

Zusätzlich mussten noch eine sogenannte Security Group erstellt werden. Diese verwaltet die Definition von Ports, welche geöffnet werden, um eine Kommunikation mit dem Server zu erlauben. In der nachfolgenden Tabelle 7 sind die Berechtigungen, welche in diesem Fall gegeben wurden, aufgelistet. Bei den ersten dreien handelt es sich um Standardwerte, welche automatisch freigeschaltet werden. Die letzten beiden wurden speziell für dieses Projekt erstellt.

Port	Typ	Verwendung im Projekt
22	SSH	Kommunikation mittels Putty mit dem Server
80	http	
443	https	
3000	TCP allgemein	Backend-Zugang
4200	TCP allgemein	Frontend-Zugang

Tabelle 7: Port-Zuweisung

2.4.3 Datenbank

Als Datenbank wird die NoSQL-Datenbank MongoDB in Version 3.4.5 verwendet [Web2]. Dem NoSQL-Konzept und wieso sein Einsatz hier sinnvoll ist, wurde ein eigenes Kapitel gewidmet (Siehe Seite 39 Exkurs NoSQL).

2.4.4 Backend

Für das Backend wird Node.js [Web3] in Version 8.0.0 mit dem Webframework Express [Web4] verwendet. Mit Node.js wird der Web Server erstellt, welcher in JavaScript programmiert wird. Dabei wird mit dem Paketmanager npm eine grosse Auswahl an Softwarebibliotheken zur Verfügung gestellt.

Folgende Schnittstellen müssen zur Verfügung gestellt werden:

- Person hinzufügen
- Person löschen
- Personen suchen
- Login

2.4.5 Frontend

Um das Frontend zu gestalten, wird das Framework Angular verwendet [Web5].

Bei Angular handelt es sich um ein Webapplikationsframework. Es verwendet TypeScript [Web6], eine Erweiterung von JavaScript. Für die Entwicklung dieser Anwendung wurde Angular in der Version 1.6.4 verwendet.

3 Applikationsentwicklung

3.1 Kenntnisaneignung

Da meine Kenntnisse der Webentwicklung vor dieser Arbeit nur beschränkt wahren, musste ich mir zuerst die Grundkenntnisse aneignen und mich mit der Materie etwas auseinandersetzten.

Dieser Abschnitt beschreibt meine persönlichen Erfahrungen.

Da es sich um eine praktisch orientiertes Projekt handelte, war ich mir von Anfang an sicher, dass ich so früh wie möglich erste Übungen machen wollte, um Erfahrungen zu sammeln. Dennoch brauchte ich zuerst einige Grundkenntnisse.

Dafür habe ich mich im Web zu den Schlagworten REST, MongoDB und Node.js etwas eingelesen, denn diese Technologien standen bereits früh fest.

Mit dem Einlesen hat sich mein Blickfeld auf Express, Angular und Web Token erweitert.

Nach dieser ersten Phase der Einarbeitung habe ich zwei YouTube-Tutorials durchgeführt [Siehe Anhang C [1],[2]]. So konnte ich mir an einem Beispiel praktische Kenntnisse aneignen.

Die Implementierungsleistung basieret auf diesen Erkenntnissen.

3.2 Entwicklungswerkzeuge

Bei der Entwicklung dieser App wurden konventionelle Werkzeuge verwendet. Da die Komplexität relativ überschaubar ist, wurde die Entwicklung mittels eines Texteditors und nicht einer grösseren Entwicklungsumgebung durchgeführt. Als Texteditor wurde Atom in der Version 1.19.0 verwendet. Dieser Texteditor beinhaltet alle notwendigen Hilfsfunktionen, ohne dabei allzu schwerfällig und kompliziert zu sein.

Für die Kommunikation zum Server wurden Putty und WinSCP verwendet.

- Bei Putty 0.67 [Web7] handelt es sich um ein Free SSH und Telnet Client für Windows. Damit konnte auf die Kommandozeile des Ubuntu-Servers zugegriffen werden.
- WinSCP 5.9.3 [Web8] ist ein Free SFTP und FTP Client für Windows, der den Austausch von Dateien ermöglicht.

Somit wurden Befehle mittels Putty ausgeführt.

Da Atom lokal auf einem Windows-Rechner installiert war und somit der Code auf dem Rechner erstellt wurde, konnten die Dateien mit WinSCP synchronisiert werden. Diese Art der

Entwicklung funktioniert gut für ein so kleines Projekt. Sobald das Projekt jedoch grösser wird, muss man die Entwicklungsmethode anpassen.

3.3 Ordnerstruktur

In diesem Abschnitt wird auf die Ordnerstruktur und das Zusammenspiel der einzelnen Elemente eingegangen. Dabei wird nicht der Code betrachtet, sondern nur die Rolle der einzelnen Ordner und Dateien.

Zu einem späteren Zeitpunkt werden einzelne Code-Abschnitte aufgezeigt und erklärt. Damit dies im Zusammenhang verstanden werden kann, wird an dieser Stelle die Dateistruktur auf dem Server thematisiert. Es wird dabei nicht detailliert auf die Inhalte der Dateien eingegangen, sondern auf deren Gruppierungen. Auch werden nicht alle Ordner und Dateien erwähnt, der Fokus liegt auf denjenigen, die zum Verständnis der Funktionsweise relevant sind.

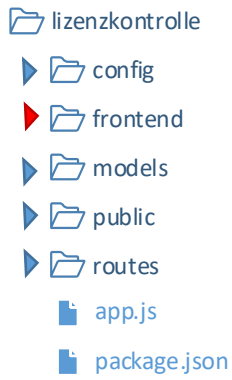


Abbildung 5 Ordnerstruktur

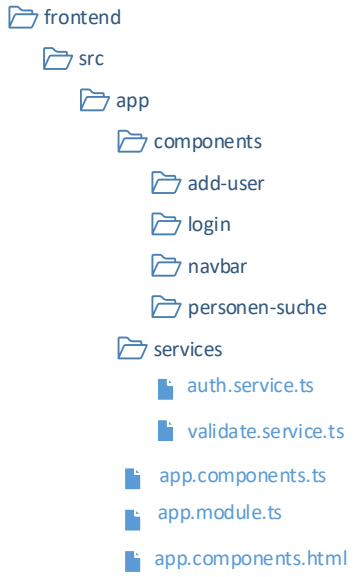
In Abbildung 5 ist die Struktur des Projektes auf erster Ebene abgebildet.

Die blau markierten Ordner bilden die Backend-Struktur und der rot markierte Ordner bildet die Frontend-Struktur.

Auf die Ordner models und routes wird im Abschnitt *Entwicklung* auf Seite 16 genauer eingegangen.

Der Ordner config wird für die Datenbankverbindung sowie für Web Tokens verwendet. Der Ordner public wird an dieser Stelle nicht verwendet, da für das Frontend ein eigener Ordner vorgesehen ist.

Weiter sind in der Abbildung zwei Dateien ersichtlich. Bei app.js handelt es sich um den Startpunkt des Backends. In der Datei package.json sind alle Abhängigkeiten der Applikation gespeichert. Dies dient dazu, dass die app transportfähig ist: Sobald app.js das erste Mal mittels Node.js gestartet wird, werden alle notwendigen Softwarepakete installiert.



Die Struktur des Frontends (Abbildung 6) wird zu einem grossen Teil bei der Erstellung eines Angular-Client-Projektes vorgegeben. Der Kommandozeilenbefehl in Abbildung 7 ist für das Erstellen des neuen Projektes zuständig.

Die applikationsspezifischen Elemente befinden sich in den Ordner components und services.

Jeder Component-Ordner stellt ein Element des Frontends dar. Dabei sind jeweils eine TypeScript- und eine HTML-Datei relevant.

Bei den Services handelt es sich um TypeScript-Dateien, die Methoden zur Verfügung stellen, welche in den Komponenten verwendet werden. Dabei handelt es sich um Methoden zur Authentifizierung des Benutzers und Methoden zur Eingabeüberprüfung.

Abbildung 6: Dateistruktur Frontend

```
ubuntu@ip-172-31-2-230:~/lizenzkontolle$ ng new frontend
```

Abbildung 7: Kommandozeilenbefehl ng new

Die drei Dateien, welche sich im Ordner app befinden, sind generische Dateien, die bei der Erstellung automatisch erstellt werden. Die Abhängigkeiten innerhalb des Frontend werden dort beschrieben. Das meiste wird automatisch beim Erstellen einer Komponente gemacht.

3.4 Entwicklung des Backends

3.4.1 REST-Schnittstellen

Nachfolgend werden in den Tabellen 8- 13 alle REST-Schnittstellen genau dokumentiert.

Anwendungsfall	Wird verwendet, um eine neue Person in der Datenbank anzulegen.
Methode	Post
Pfad	/register
Header	- Content-Type: application/json - Authorization: token
Body	{ "firstName":"input", "lastName":" input ",

	"email": " input ", "sjvNr":" input ", "role":" input ", "lizenz":" input ", "password":" input ", "geburtstag":" input" }
Response	Registrierung erfolgreich oder nicht

Tabelle 8: Schnittstelle «Personenregistrierung»

Anwendungsfass	Wird verwendet, um eine neue Person in der Datenbank anzulegen. Im Gegensatz zum ersten Fall wird hier eine Person ohne Passwort angelegt.
Methode	Post
Pfad	/register2
Header	- Content-Type: application/json - Authorization: token

Body	{
	"firstName":"input",
	"lastName":" input ",
	"email": " input ",
	"sjvNr":" input ",
	"role":" input ",
	"lizenz":" input ",
	"geburtstag":" input"
	}

Response	Registrierung erfolgreich oder nicht

Tabelle 9: Schnittstelle «Personenregistrierung zwei»

Anwendungsfall	Wird zum Login verwendet.
Methode	Post
Pfad	/authenticate
Header	Content-Type: application/json

Body	{
	"email":"input",
	"password":"input"
	}

Response	Bei Misserfolg: wrong password Bei Erfolg: Token und die Nutzerdaten

Tabelle 10: Schnittstelle «Login»

Anwendungsfall	Wird verwendet, um eine Person zu finden, deren Name die mitgegebenen Buchstaben enthält.
Methode	Get
Pfad	/lizenzierte/*suche*
Header	Authorization: token
Body	
Response	Bei Misserfolg: keine Person gefunden Bei Erfolg: Liste der Personenobjekte, welche die Eingabebuchstaben an Stelle «suchen» enthalten

Tabelle 11: Schnittstelle «Personensuche»

Anwendungsfall	Wird verwendet, um eine Person zu löschen.
Methode	Delete
Pfad	/lizenziert/*id*
Header	Authorization: token
Body	
Response	Person gelöscht

Tabelle 12: Schnittstelle «Person löschen»

Anwendungsfall	Zeigt die gesamten Nutzerdaten an.
Methode	get
Pfad	/profile
Header	Authorization: token
Body	
Response	Nutzerdaten

Tabelle 13: Schnittstelle «Profil anzeigen»

3.4.2 Interessante Aspekte im Code

Im Bereich des Backends wird im Folgenden auf zwei Dateien eingegangen: user.js und users.js. In der Datei users.js sind alle Routes definiert. Alle REST-Methoden sind dort implementiert. Sie rufen dann eine Methode, auf welche die Datenbankabfragen ausführt. Die Methoden für Datenbankabfragen befinden sich in der Datei user.js. Zusätzlich zu diesen Methoden ist dort auch das Datenbankschema definiert.

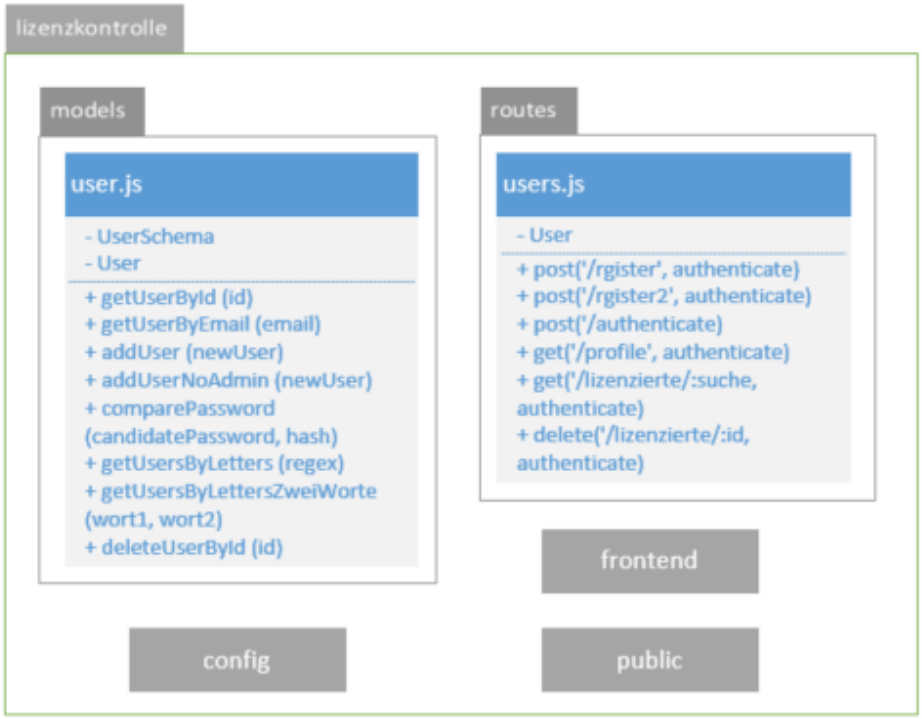


Abbildung 8: Klassendiagramm Backend

Nachfolgend werden nun einige Elemente genauer betrachtet. Es wird immer eine REST-Methode und die dazugehörige Datenbankaufrufmethode in user.js diskutiert.

Lizenzierte suchen

Dabei handelt es sich um die Schnittstelle, um anzuzeigen, welche Personen lizensiert sind.

Dafür wird eine GET-Methode verwendet, da es keinen neuen Datenbankeintrag gibt, sondern nur aus der Datenbank gelesen wird. In Abbildung 9 ist der Anfang dieser Methode dargestellt. Dort sieht man auch die Aufteilung des Suchbegriffes in zwei Worte, falls ein Leerschlag enthalten ist.

```
106   router.get('/lizenzierte/:suche',  passport.authenticate('jwt', {session:false}), (req, res, next) => {
107
108     var suche = req.params.suche;
109     var regex = new RegExp(suche,"i");
110
111     var sucheMerereWorte = suche.split(" ");
112     var wort1 = new RegExp(sucheMerereWorte[0],"i")
113     var wort2 = new RegExp(sucheMerereWorte[1],"i")
114
```

Abbildung 9: Ausschnitt aus users.js

Die Eingabe, die vom Frontend kommt, wird an dieser Stelle nicht im Body übermittelt, sondern direkt als Methodenparameter. Die Eingabe ist ein String, welcher einen Leerschlag beinhalten kann. Falls dies der Fall ist, muss er als zwei Worte interpretiert werden, was die Suche etwas anders gestaltet. Darauf wird an späterer Stelle nochmals eingegangen.

Weiterhin ist es wichtig, dass Gross- und Kleinschreibung nicht beachtet werden soll. Dazu ist bei der Definition des regulären Ausdrucks ein separater Parameter i erforderlich.

Wie bereits erwähnt, gestaltet sich die Datenbankabfrage in der Datei user.js unterschiedlich, abhängig davon, ob ein Wort oder zwei Worte eingegeben wurden. In Abbildung 10 ist zu sehen, wir unterschieden wird, ob ein oder zwei Worte als Eingabe angegeben wurden.

```
115   // wenn kein Lehrschlag eingegben wurde
116     if(sucheMerereWorte.length<2){
117       User.getUsersByLetters(regex, (err, personen) => {
118         if(err) throw err;
119         if(!personen){
120           return res.json({success: false, msg: 'keine Person vorhanden'});
121         }
122         if(personen){
123           return res.json(personen);
124         }
125       });
126     } else {
127       User.getUsersByLettersZweiWorte(wort1, wort2,  (err, personen) => {
128         if(err) throw err;
129         if(!personen){
130           return res.json({success: false, msg: 'keine Person vorhanden'});
131         }
132         if(personen){
133           return res.json(personen);
134         }
135       });
136     }
137   });
```

Abbildung 10: Ausschnitt aus user.js

Bei einem Wort wird in den Feldern Vorname, Nachname und SJV-Nummer gesucht. Es kann auch direkt die SJV-Nummer eingegeben werden.

Falls die Eingabe jedoch zwei Worte umfasst, stellen diese Vor- und Nachname dar. Somit muss in diesen zwei Felder gesucht werden. Wichtig ist, dass unbekannt ist, ob zuerst der Vor- oder Nachname eingegeben wurde. Aus diesem Grund müssen beide Kombinationen überprüft werden.

Registrierung

Bei der Registrierung gibt es, ähnlich zur Personensuche, auch zwei Szenarien je nach Eingabedaten. Die Umsetzung ist hier aber etwas anders. Die Unterscheidung findet im Frontend statt. Abhängig davon, ob jemand ein Passwort hat oder nicht, wird eine bestimmte Post-Methode aufgerufen. In Abbildung 11 ist die Methode abgebildet, die aufgerufen wird, falls ein Nutzer ein Passwort hat und somit über Zugangsrechte verfügt. Der einzige Unterschied zwischen Aufrufen mit und ohne Passwort besteht darin, dass eine andere Methode aufgerufen, wird um den Nutzer hinzuzufügen.

```
9   // Registrierung
10  // um neue personen einzufügen
11  router.post('/register',  passport.authenticate('jwt', {session:false}), (req, res, next) => {
12
13    "use strict";
14    let newUser = new User ({
15      firstName: req.body.firstName,
16      lastName: req.body.lastName,
17      email: req.body.email,
18      sjvNr: req.body.sjvNr,
19      role: req.body.role,
20      password: req.body.password,
21      lizenz: req.body.lizenz,
22      geburtstag: req.body.geburtstag
23    });
24
25    User.addUser(newUser, (err, user) => {
26      if(err){
27        res.json({success: false, msg: 'registrierung nicht erfogreich'});
28      } else {
29        res.json({success: true, msg: 'registrierung erfolgreich'});
30      }
31    });
32
33  });
```

Abbildung 11: Ausschnitt aus users.js

Wie gerade angesprochen, wird ein Post-Aufruf verwendet, da Daten in die Datenbank geschrieben werden.

Der REST-Anfrage wird ein Body mit dem Nutzerobjekt angehängt. Dies beinhaltet alle Angaben zu der einzutragenden Person im JSON-Format.

In der user.js Datei werden dann alle Felder in der Datenbank gespeichert.

```
56    module.exports.addUser = function(newUser, callback){
57      bcrypt.genSalt(10, (err, salt) => {
58        bcrypt.hash(newUser.password, salt, (err, hash) => {
59          if(err) throw err;
60          newUser.password = hash;
61          newUser.save(callback);
62        });
63      });
64    }
65
66    module.exports.addUserNoAdmin = function(newUser, callback){
67          newUser.save(callback);
68    }
69
```

Abbildung 12: Ausschnitt aus user.js

Wenn man wie in Abbildung 12 beide Methoden gegenüberstellt, ist nur ein kleiner Unterschied ersichtlich: Falls ein Passwort eingegeben wurde, wird auf dieses zuerst eine Hash-Funktion angewendet, bevor es wieder im Objekt gespeichert wird. Der nächste Schritt – das Speichern in der Datenbank – ist identisch implementiert.

Authentifizierung

Die Authentifizierung bzw. der Login erfolgt in einer Post-Methode. Der Body enthält dabei E-Mail-Adresse und Passwort.

Diese Methode macht etwas mehr als nur eine Datenbankabfrage, wie in Abbildung 13 ersichtlich, die den ersten Teil dieser Methode darstellt. Zuerst wird anhand der E-Mail-Adresse die Person gesucht. Anschliessend wird der Hash-Wert des gesendeten Passwortes mit dem in der Datenbank gespeicherten Wert verglichen.

```
60    // authentifizeiung
61    // für login
62    router.post('/authenticate', (req, res, next) => {
63      const email = req.body.email;
64      const password = req.body.password;
65
66      User.getUserByEmail(email, (err, user) => {
67        if(err) throw err;
68        if(!user){
69          return res.json({success: false, msg: 'User not found'});
70        }
71
72        User.comparePassword(password, user.password, (err, isMatch) => {
73          if(err) throw err;
74          if(isMatch){
75            const token = jwt.sign(user, config.secret, {
76              expiresIn: 604800 // Lauft nach einer woche ab
77            });
```

Abbildung 13: Ausschnitt aus users.js

Falls die Passwörter übereinstimmen, wird für die Authentifizierung ein Web Token im JSON-Format benutzt. Dabei kann konfiguriert werden, wie lange dies gültig sein. In der vorgestellten Applikation ist dieser Wert auf einer Woche gesetzt, er basierend auf Erfahrungswerten auch noch angepasst werden.

Falls die zwei Hash-Werte übereinstimmen, werden dem Benutzer drei Komponenten zurückgesendet: eine Bestätigung, dass der Login erfolgreich war, das Web Token und die Benutzerdaten. Bei den Benutzerdaten ist zu beachten, dass nicht das ganze Nutzerobjekt

gesendet wird. Es wird stattdessen ein neues Objekt erstellt, welches den Hash-Wert des Passwortes nicht enthält – dieser soll dem Frontend nicht zur Verfügung stehen.

Im zweiten Teil der Post-Methode, welcher in der Abbildung 14 dargestellt ist, ist ersichtlich, dass ein neues Objekt zurückgegeben wird.

```
78    //neues objekt wird kreiert weil wir nicht user schikken wollen dort ist das hash drinnen
79            res.json({
80              success: true,
81              token: 'JWT '+token,
82              user: {
83                id: user._id,
84                firstName: user.firstName,
85                lastName: user.lastName,
86                role: user.role,
87                geburtstag: user.geburtstag
88              }
89            });
90          } else {
91            return res.json({success: false, msg: 'Wrong password'});
92          }
93        });
94      });
95    });
96
```

Abbildung 14: Ausschnitt aus users.js

Person löschen

Damit Personen in der Datenbank auch wieder gelöscht werden können, wurde eine Delete-Methode erstellt.

Diese Methode ist eine der einfachsten umgesetzten Methoden. Sie ist, wie alle anderen Methoden, geschützt, damit nur Berechtigte darauf zugreifen können. Als Eingabe wird vom Frontend die ID der Person angegeben, welche gelöscht werden soll. Als Rückgabe erfolgt nur eine Mitteilung, ob das Löschen funktioniert hat oder nicht. Die ID, wird an Methode weitergegeben, welche für das Löschen zuständig ist, wie in Abbildung 15 dargestellt.

```
140   router.delete('/lizenzierte/:id', passport.authenticate('jwt', {session:false}), (req, res, next) =>{
141       var id = req.params.id;
142       User.deleteUserById(id,(err, personen) => {
143
144         if(err) throw err;
145         if(personen){
146           return  res.json({success: true, msg: 'Person gelöscht'});
147         }
148       });
149     });
150
```

Abbildung 15: Ausschnitt aus users.js

Um die Person anhand der ID zu löschen, wird die Funktion findOneAndRemove verwendet, wie in Abbildung 16 zu sehen.

```
101    module.exports.deleteUserById = function(id,callback){
102      User.findOneAndRemove({"_id": id}, function(err, docs){
103        callback(err,docs)
104      });
105    }
106
```

Abbildung 16: Ausschnitt aus user.js

3.4.3 Testing mit Postman

Da für das Testen der REST-Schnittstellen eine separate Software verwendet wurde, ist dieser eine eigener Abschnitt gewidmet.

Beim Testen der Schnittstellen war es wichtig, sicherzustellen, dass die Daten richtig verarbeitet werden, bei fehlenden Eingabewerten sinnvolle Fehlermeldungen zurückgegeben werden und alle notwendigen Schnittstellen richtig geschützt sind.

Es wäre denkbar gewesen, das Frontend zu entwickeln und anschliessend mit diesem zu prüfen, ob es Fehler hat. Es wäre jedoch viel aufwendiger gewesen, diese Fehler zu finden. Aus diesem Grund wurde Postman zum Testen verwendet. Dabei handelt es sich um eine API-Entwicklungsumgebung, die in dieser Arbeit für das Testen der REST-Schnittstellen verwendet wurde. Dabei können auf einfache Art sowohl Header definiert werden, als auch der Body und somit jegliche REST-Methoden getestet werden.

Am Beispiel für das Eintragen einer neuen Person wird in der Abbildung 17 aufgezeigt, wie die Schnittstelle getestet werden kann.

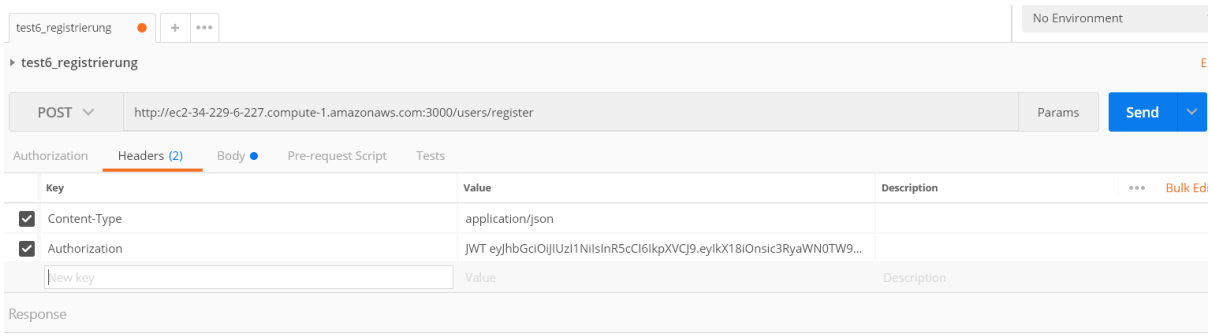


Abbildung 17: Ansicht Postman Headers

Auf dieser Benutzeroberfläche kann alles parametrisiert werden, wie es bei einem Frontend auch gemacht würde. Die Methode kann gewählt und anschliessend der Pfad angegeben werden.

Anschliessend kann man die Header wählen. Wie man hier sehen kann, werden zwei Header verwendet: eine Angabe, dass ein JSON-Inhalt gesendet wird und ein Autorisierungsobjekt – hier wird das Web Token mitgegeben. Ohne diese Autorisierung entstünde eine Fehlermeldung, da dieser Pfad im Backend geschützt ist.

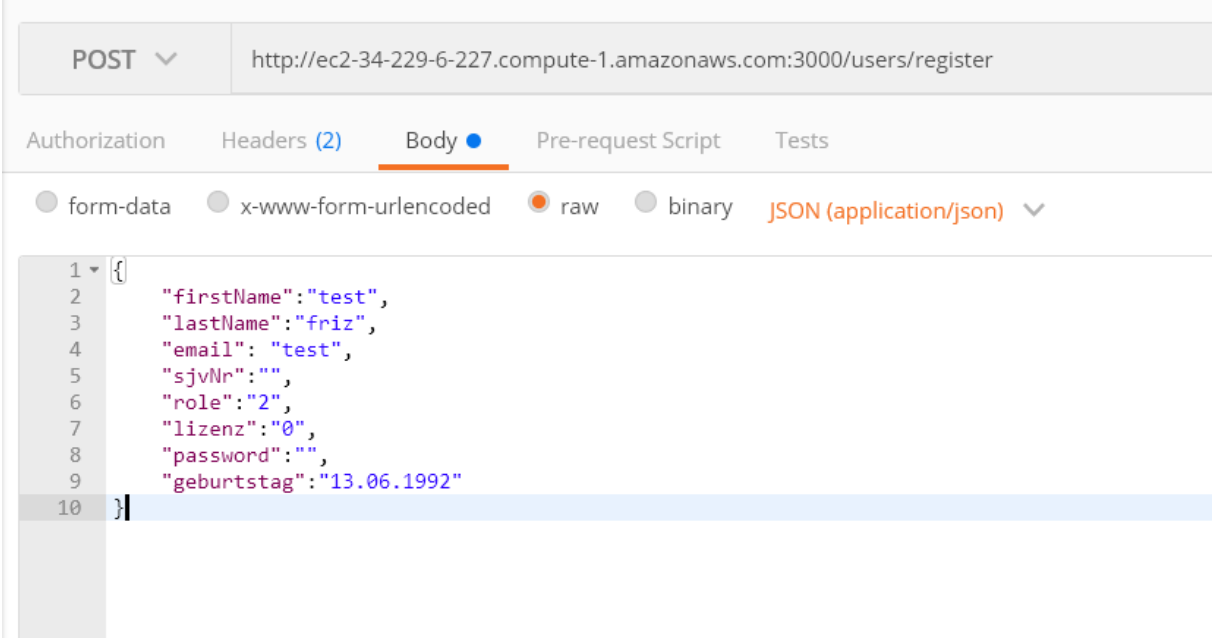


Abbildung 18: Ansicht Postman Body

In einem nächsten Schritt kann im JSON-Format der Inhalt mitgegeben werden, welcher in der Datenbank gespeichert wird. Sobald man diesen absendet, erhält man eine Antwort im JSON-Format, die angibt, ob die Operation erfolgreich war sowie eine Nachricht, welche im Backend definiert ist.

Auf diese Art können alle Schnittstellen getestet werden. Dieser Prozess gibt die Sicherheit, dass, falls es Probleme im Frontend gibt, sich diese Fehler wirklich auf das Frontend beschränken und nicht möglicherweise auch noch im Backend existieren.

3.5 Entwicklung des Frontends

3.5.1 Interessante Aspekte im Code

Das Frontend umfasst etwas komplexere Dateistrukturen als das Backend. Es existiert zu jeder Komponente sowohl eine HTML- als auch eine TypeScript-Datei, welche für das Projekt angepasst wurde. Zusätzlich gibt es zu jeder Komponente noch zwei weitere Dateien: eine CSS-Datei und eine weitere TypeScript-Datei, welche aber nicht verändert wurde. Diese vier Dateien entstehen bei dem Erstellen einer Komponente über die Kommandozeile (Siehe Abbildung 19).

```
ubuntu@ip-172-31-2-230:~/lizenzkontolle/frontend/src/app/components$ ng g component navbar
```

Abbildung 19: Kommandozeilenbefehl ng g componet

Zusätzlich zu diesen komponentenbezogenen Elemente gibt es noch zwei weitere Dateien im Ordner Services. Diese stellen Dienste dar und werden, wie in Abbildung 20 zu sehen, über einen gesonderten Befehl erstellt.

```
ubuntu@ip-172-31-2-230:~/lizenzkontolle/frontend/src/app/components$ ng g service auth
```

Abbildung 20: Kommandozeilenbefehl ng g service

Dabei werden die Dateien auth.services.ts und validate.services.ts erzeugt. Die erste Datei beinhaltet alle Methoden, welche auf die REST-Schnittstellen zugreifen. In der zweiten Datei werden Nutzereingaben auf deren Richtigkeit überprüft.

Da das Layout bei dieser Arbeit nicht im Vordergrund stand, sondern die Funktionalitäten, wurde am CSS-Dokument gar nichts verändert und das HTML-Dokument ist sehr schlicht gehalten.

Person registrieren

Die Personenregistrierung erfolgt über ein gewöhnliches HTML-Formular. Beim Absenden des Formulars wird eine Methode aufgerufen, welche wiederum weitere Methoden aufruft, welche den Inhalt der Formularfelder anhand verschiedener Kriterien überprüfen. Diese werden im Folgenden der Reihe nach erläutert.

Der vollständiger Code der Methode befindet sich in Anhang A Source Code

- validateRoleAdmin (Abbildung 21)
  In dieser Methode wird geprüft, ob die Person die Berechtigung, hat eine neue Registrierung hinzuzufügen. Eine Person, welche die Rechte zum Hinzufügen nicht hat, kann den Link für auf diese Seite gar nicht sehen. Dennoch wäre es denkbar, dass die Person durch Eingabe des richtigen Pfades auf die Seite kommt. Aufgrund dieser Methode kann die Person dann dennoch niemanden hinzufügen.

```
50    validateRoleAdmin(){
51      var user = JSON.parse(localStorage.getItem('user'))
52
53      if( user.role == "2"){
54        return true;
55      } else {
56        return false;
57      }
58    }
59
```

Abbildung 21: Ausschnitt aus validate.service.ts

- ValidateRegister (Abbildung 22)
  Diese Funktion prüft, ob die erforderlichen, mit einem Stern markierten Felder ausgefüllt sind. Dabei ist wichtig, sowohl auf «undefined», wie auf leere Strings zu

überprüfen, denn sobald etwas eingetragen und dann wieder gelöscht wird, enthält der Wert nicht mehr «undefined», sondern einen leeren String.

```
9     validateRegister(user){
10      if(user.firstName == undefined ||
11         user.lastName == undefined ||
12         user.role == undefined ||
13         user.lizenz == undefined ||
14         user.firstName == "" ||
15          user.lastName == "" ||
16          user.role == "" ||
17          user.lizenz == "" ){
18        return false;
19      } else {
20        return true;
21      }
22    }
23
```

Abbildung 22: Ausschnitt aus validate.service.ts

- ValidateGeburtstag (Abbildung 23)
  Diese Funktion wird nur aufgerufen, wenn ein Wert enthalten ist. Wie der Name sagt, überprüft diese Funktion, ob die Form des Datums korrekt ist. Es werden zwei Formen akzeptiert (dd.mm.jjjj / dd-mm-jjjj). Falls die Form falsch ist, wird dem Anwender mittels einer Flash Message angezeigt, wie eine gültige Form aussehen sollte.

```
24    validateLogicGeburtstag(user){
25
26      const re = /^\d{2}([./-])\d{2}\1\d{4}$/;
27     return re.test(user.geburtstag);
28    }
29
```

Abbildung 23: Ausschnitt aus validate.service.ts

- ValidateLogicPassword (Abbildung 24)
  In dieser Methode wird überprüft, dass es bei den Zugangsrechten keine Widersprüche gibt: Wenn jemand ein Zugangsrecht hat, dann muss diese Person auch eine E-Mail-Adresse und ein Passwort haben.

```
30    validateLogicPassword(user){
31      if((user.role == "1" || user.role =="2")&&
32        (user.password == undefined || user.email == undefined || user.password == "" || user.email == "")){
33        return false;
34      } else {
35        return true;
36      }
37
```

Abbildung 24: Ausschnitt aus validate.service.ts

- validateLogicLicense (Abbildung 25)
  Auch hier wird der Zusammenhang von mehreren Feldern überprüft. Denn wenn

jemand lizenziert ist, muss diese Person auch eine SJV-Nummer haben. Umgekehrt gilt dieser Zusammenhang nicht, denn die SJV-Nummer ist ein Leben lang gültig, wohingegen die Lizenz nur ein Jahr lang gültig ist.

```
40    validateLogicLicense(user){
41      if((user.lizenz == "1" )&&
42        (user.sjvNr == undefined ||  user.sjvNr == "")){
43        return false;
44      } else {
45        return true;
46      }
47
48    }
49
```

Abbildung 25: Ausschnitt aus validate.service.ts

- registerUser (Abbildung 26)
  Diese Methode wird aufgerufen, falls vorher kein Fehler zurückgegeben wurde. Die Methode dient dazu, auf die REST-Schnittstelle zuzugreifen und somit die Person in der Datenbank zu speichern.

```
13      registerUser(user){
14        let headers = new Headers();
15        this.loadToken();
16        headers.append('Authorization', this.authToken); // tocken wird in den heder gegeben
17        headers.append('Content-Type','application/json');
18        if((user.password == undefined)||(user.password == "") ){
19
20          return this.http.post('http://ec2-34-229-6-227.compute-1.amazonaws.com:3000/users/register2',
21          user,{headers: headers})
22            .map(res => res.json());
23        } else {
24
25          return this.http.post('http://ec2-34-229-6-227.compute-1.amazonaws.com:3000/users/register',
26          user,{headers: headers})
27            .map(res => res.json());
28        }
29
30      }
31
```

Abbildung 26: Ausschnitt aus auth.service.ts

Wie man feststellen kann, wird die E-Mail-Adresse nicht auf ihre Form geprüft. Dies mag erstaunen, ist jedoch durch eine Vorgabe des SJV begründet: Es gibt Standard-Administrations-Logins, welche keine gültige E-Mail-Adresse als Nutzernamen haben. Somit ist es also auch möglich, einen anderen String als Benutzernamen zu verwenden.

Personensuche

Die Personen suche erfolgt auf der Startseite. Dies hat den Vorteil, dass Nutzer mit gültigem Web Token im Browser-Speicher direkt die Suche verwenden können. Falls eine Person noch kein Web Token gespeichert hat, wird sie auf die Login-Seite weitergeleitet.

Der Button «suche» hat eigentlich keine Funktion, er dient nur dazu, den Nutzern die Sicherheit zu geben, dass wirklich gesucht wurde. Dies beruht auf Erfahrungswerten, die aussagen, dass es für manche Personen einfacher zu verstehen ist, wenn sie für die Suche einen Knopf drücken können.

Sobald drei Zeichen eingegeben wurden, werden alle Personen, deren Namen diese Zeichenketten enthalten, angezeigt. Dies geschieht auch ohne Betätigung des Suchknopfs.

In Abbildung 27 ist zu sehen, dass bei einer Eingabe immer überprüft wird, ob drei oder mehr Buchstaben eingegeben wurden. Erst dann wird die nächste Methode aufgerufen.

```
42      whenTyping() {
43        if (this.suche.length >= 3) {
44          this.personenSuchenOhneMeldung();
45        }
46      }
47
```

Abbildung 27: Ausschnitt aus personen-suche.component.ts

Die Beschränkung, das Suchergebnisse erst bei drei mindestens Zeichen anzeigt werden, wurde umgesetzt, da ansonsten bei der Suche nach nur einem Buchstaben sehr viel angezeigt würde. Dies würde, gerade bei mobilen Anwendungen, zu Verzögerungen führen.

Das Resultat, das von dem Backend zurückgesendet wird, ist eine Liste von Objekten die den verschiedenen Nutzern entsprechen. Mittels des Konstruktes *ngFor kann die Liste der Objekte durchgegangen und die gewünschten Felder aufgelistet werden. Zusätzlich kann mit dem Bezeichner ngif angeben werden, ob eine Person eine Lizenz hat oder nicht, «0» entspricht nicht lizenziert; «1» entspricht lizenziert. Die beiden Bedingungen sind in Abbildung 28 zu sehen.

```
22          <tr *ngFor="let person of personen">
23             <td>{{person.firstName}}</td>
24            <td>{{person.lastName}}</td>
25            <td>{{person.geburtstag}}</td>
26            <td>{{person.sjvNr}}</td>
27            <td *ngIf="person.lizenz === '1'">Lizenziert</td>
28            <td *ngIf="person.lizenz === '0'">nicht Lizenziert</td>
```

Abbildung 28: Ausschnitt aus personen-suche.component.html

Personen löschen

Die erste Umsetzungsidee war, eine neue Seite zum Löschen von Personen zu erstellen, auf die nur die Administratoren zugreifen können. Dafür hätte man jedoch die Personensuche ein zweites Mal implementieren müssen, was wenig sinnvoll ist. Somit besteht die jetzige Lösung darrin, dass der Button zum Löschen der Personen für Administratoren sichtbar ist.

In der Variable isAdmin wird bereits mit der Methode ngOnInit beim Aufbau der Seite festgelegt, ob der Benutzer ein Administrator ist oder nicht.

```
29              <td *ngIf="isAdmin">
30                  <button class="btn btn-danger" (click)="personenLoeschen(person._id)">
31                    Person Löschen
32                  </button>
33              </td>
```

Abbildung 29: Ausschnitt aus personen-suche.component.html

Mit der ngif-Bedingung, dargestellt in Abbildung 29, wird anschliessend bei den Auflistungen der Person im HTML-Dokument der Button zum Löschen angezeigt, falls die verwendende Person ein Administrator ist.

Abmelden / Logout

Der Logout-Vorgang ist ausschliesslich im Frontend umgesetzt, denn dafür werden keine Angaben aus dem Backend benötigt. In Abbildung 30 ist die Methode dargestellt, welche aufgerufen wird, sobald der Menüpunkt «Logout» betätigt wird.

```
51    onLogoutClick(){
52
53      this.authService.logout();
54      this._flashMessagesService.show('Sie sind ausgeloggd', { cssClass: 'alert-success', timeout: 2000 });
55      this.hasRole = this.validateService.validateRoleVorhanden();
56
57      this.router.navigate(['/login']);
58      return false;
59    }
```

Abbildung 30: Ausschnitt aus navbar.component.ts

Dabei wird die Methode logout aufgerufen. Diese ist in Abbildung 31 zu sehen. Dort kann man sehen, dass die Variablen zum Abmelden auf den Wert null gesetzt werden und der Browserspeicher geleert wird.

```
74      logout(){
75      this.authToken = null;
76      this.user = null;
77      localStorage.clear();
78      }
79
80    }
```

Abbildung 31: Ausschnitt aus auth.service.ts

Nachdem der Browser-Speicher geleert wurde, wird die Methode validateRolleVorhanden aufgerufen, um das Feld hasRole richtig auszufüllen.

Mit diesem letzten Schritt sind alle Werte so belegt, wie am Anfang (vor dem Login) und der Logout-Vorgang war erfolgreich.

4 Endprodukt

4.1 Benutzeroberfläche

Die Benutzeroberfläche wurde schlicht und funktional gehalten, um so eine einfache und intuitive Handhabung zu gewährleisten.

Dies ist in Abbildung 32 auf der Login-Seite zu sehen.

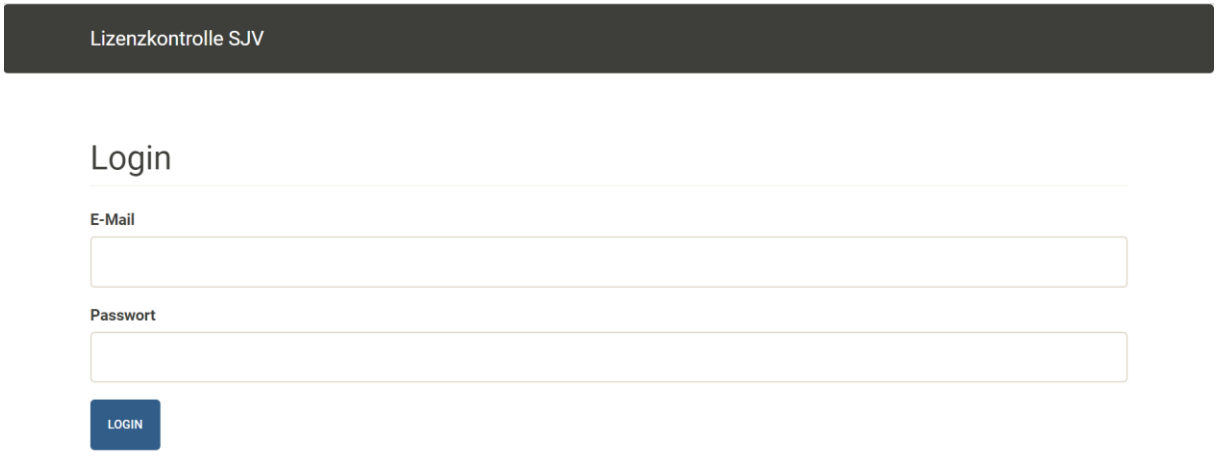


Abbildung 32: Login-Ansicht

Falls der Nutzer nur Leserechte hat, wird ihm die in Abbildung 33 gezeigte Seite dargestellt. Dabei ist wichtig, dass alle Menübereiche, auf die er keinen Zugang hat, nicht angezeigt werden.

Dem Benutzer steht nun ein Textfeld zur Verfügung, in das er seine Suchbegriffe eingeben kann. Diese Suchbegriffe können entweder eine SJV-Nummer oder einen Namen enthalten.

Falls der gesamte Name eingegeben wurde und niemand angezeigt wird, wird erst bei Betätigung des Knopfes «Suche» eine Meldung ausgegeben, die besagt, dass die Person nicht gefunden wurde. Diese Meldung wird bewusst erst zu diesem Zeitpunkt gesendet, da nicht bei jedem versehentlichen Tippfehler eine Meldung angezeigt werden sollte. Dies soll erst geschehen, sondern erst wenn der Benutzer mit der Eingabe fertig ist.

Die Personen werden immer mit den Feldern Vorname, Name, Geburtstag, SJV-Nummer sowie einer Angabe, ob sie lizenziert sind, angezeigt. Die ersten vier Felder dienen dazu, die Person identifizieren zu können. Eigentlich ist die SJV-Nummer zwar eindeutig ist und würde somit ausreichen, jedoch kennen diese nur weniger Vereinsmitglieder auswendig. Mit Name, Vorname und Geburtsdatum sollte aber gewährleistet sein, dass jede person eindeutig identifiziert werden kann. Die letzte Spalte gibt an, ob eine Person eine gültige Lizenz hat.

Die meisten Personen in der Datenbank haben eine Lizenz. Dennoch kann man nicht davon ausgehen, dass alle angezeigten Personen Lizenzen besitzen. Denn es wird mit grosser Wahrscheinlichkeit Administratoren geben, welche keine gültige Lizenz besitzen.

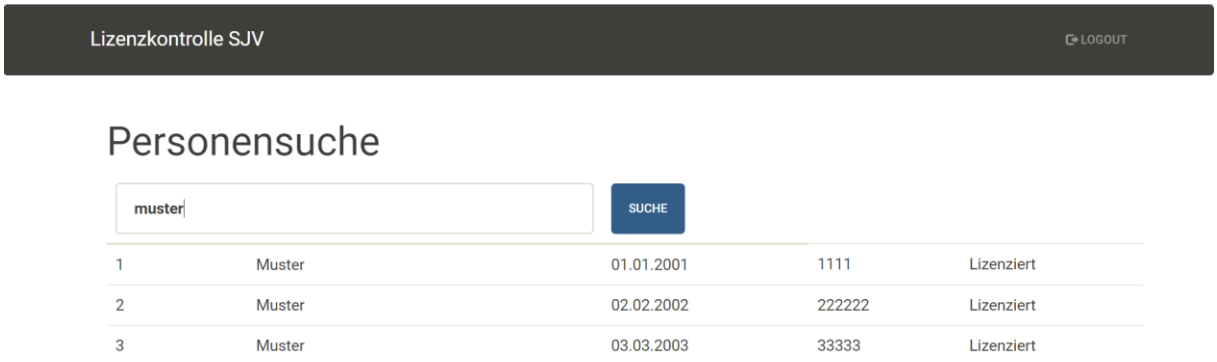


Abbildung 33: Ansicht Personensuche

Im Gegensatz zu der Seite aus Sicht eines normalen Nutzers sieht die Anwendung in der Rolle eines Administrators aus, wie in Abbildung 34 dargestellt.

Dem Administrator stehen mehr Möglichkeiten zur Verfügung. Zusätzlich zur Personensuche, welche im vorherigen Abschnitt beschrieben wurde, kann er jede Person, welche ihm angezeigt wird, durch Betätigung des Löschknopfes aus der Datenbank entfernen. Nach jedem Löschvorgang wird dem Nutzer mitgeteilt, ob er funktioniert hat.

Eine Besonderheit dabei ist, dass der Administrator sich selber nicht löschen kann. Falls er dies versucht, wird er darauf hingewiesen, dass dies nicht möglich ist.

Zusätzlich steht dem Administrator auch noch ein weiterer Menüpunkt «Personen hinzufügen» zur Verfügung.

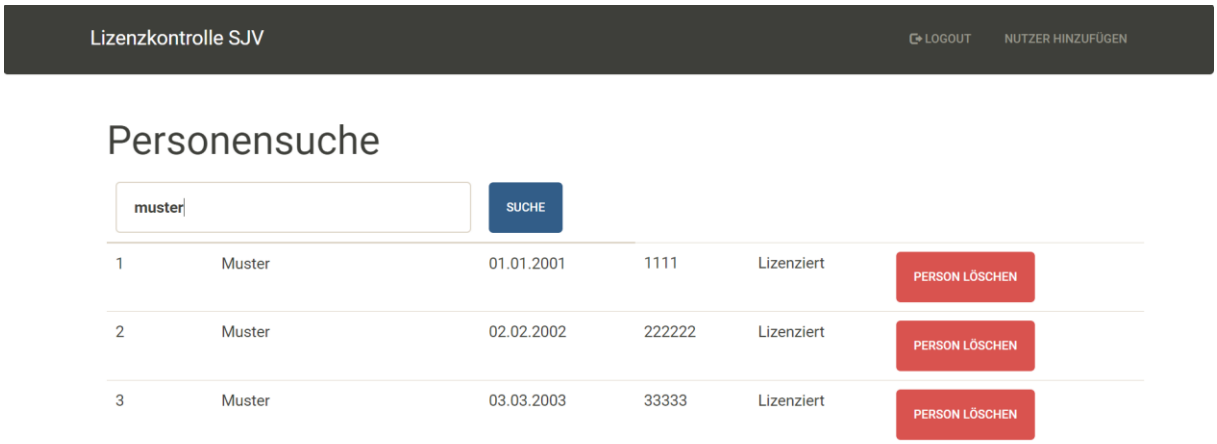


Abbildung 34: Ansicht Personensuche Admin

Falls der Administrator sich auf diese Seite zum Hinzufügen eines Nutzers begibt, erwartet ihn das in Abbildung 35 dargestellte Fenster.

Es enthält ein Webformular mit Textfeldern und Radio Buttons. Die Felder, welche mit einem Stern markiert sind, sind Pflichtfelder. Bei den anderen Feldern handelt es sich um fakultative Angaben. Abhängig davon, was ausgefüllt ist, müssen auch andere Felder ausgefüllt werden: Beispielsweise ist es nicht sinnvoll, jemandem Zugriffsrechte zu geben, aber kein Passwort. Da diese Logik für Nutzer nicht immer intuitiv ist, gibt es Nachrichten, welche beim Absenden des Formulars angezeigt werden und auf solche Eingabefehler hinweisen.

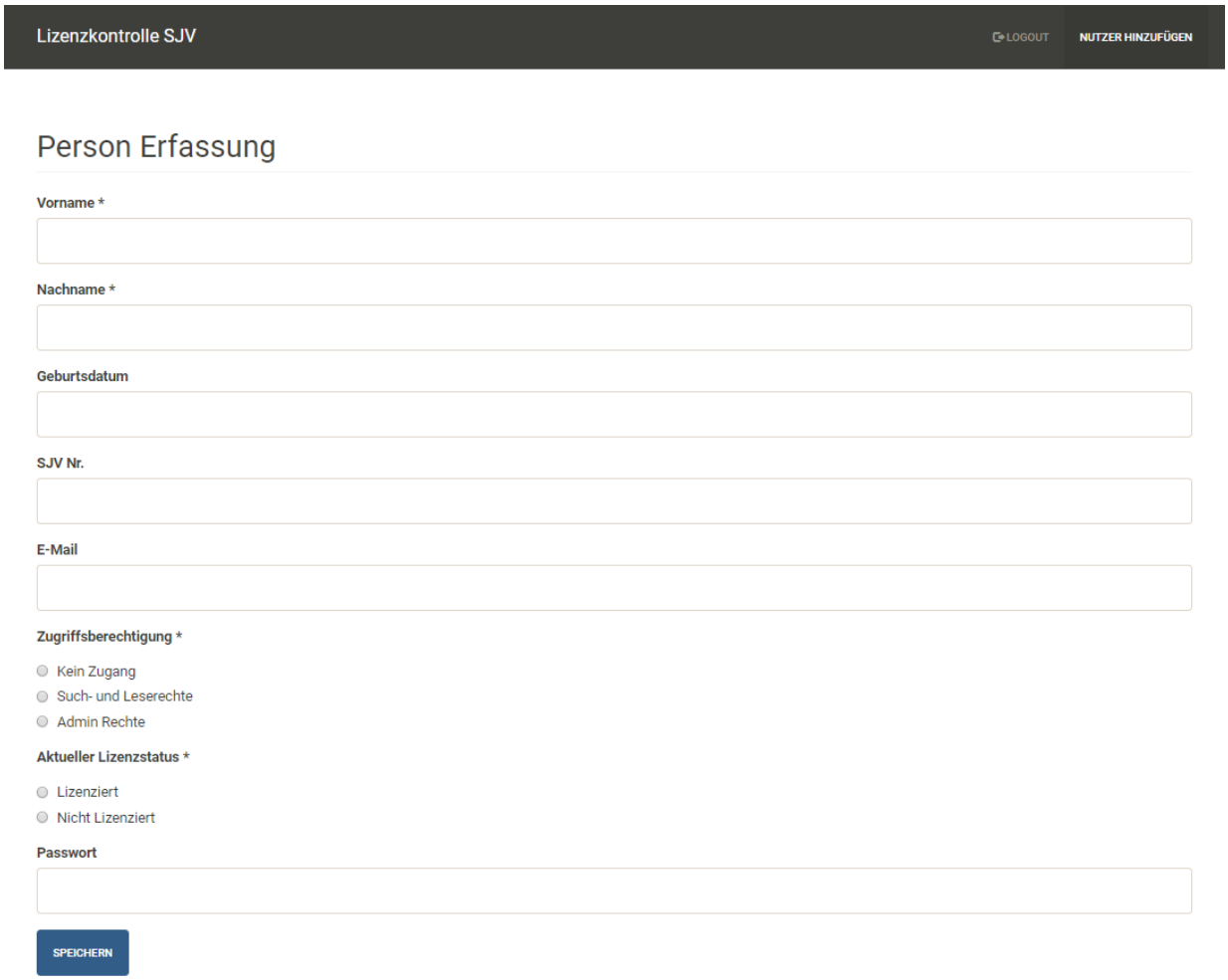


Abbildung 35: Ansicht zur Personenerfassung

In der gesamten Applikation befinden sich bewusst nur wenige Texte und Hilfestellungen. Dies soll der Übersichtlichkeit dienen. Sobald eine Fehlverwendung vorliegt wird, werden dem Benutzer sinnvolle Hinweise angezeigt.

4.2 Anwendungs-Lifecycle

Damit die Anwendung ihren Zweck erfüllen kann, muss sie über aktuelle Daten verfügen und den Administratoren den Zugang erlauben.

Der Anwendungs-Lifecycle, welcher in der Abbildung 36 dargestellt ist, wird unterteilt in eine Anwendungsinitialisierungsphase, welche einmal abläuft und eine iterative, wiederholte Phase der Jahresnutzung.

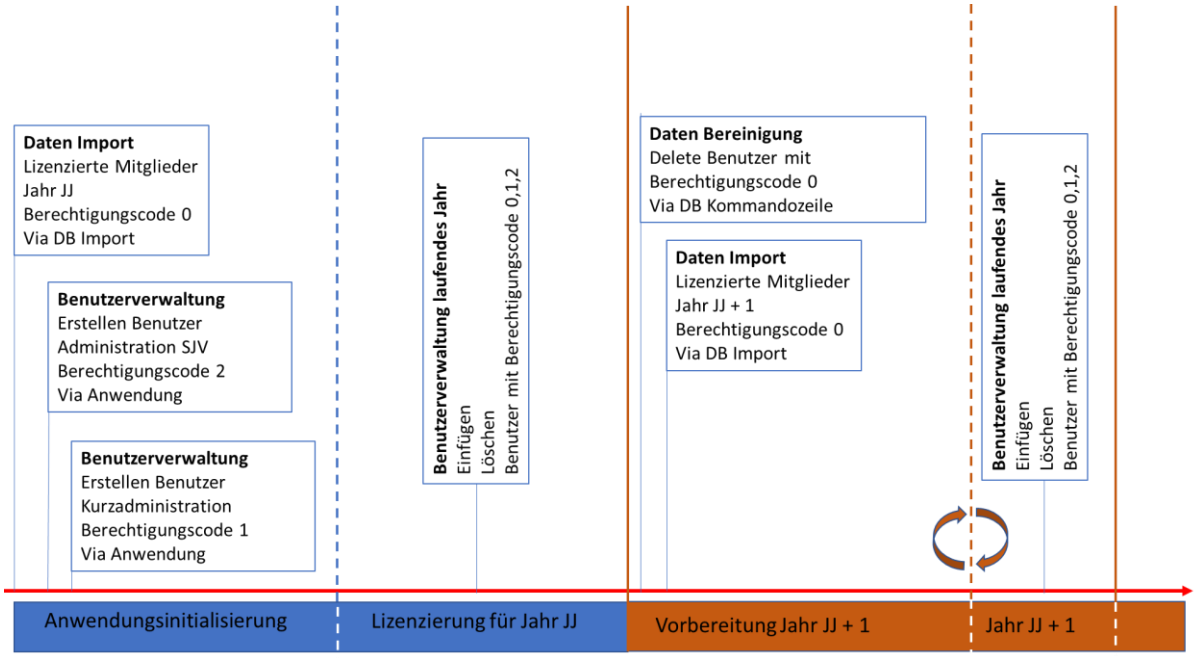


Abbildung 36: Anwendungs-Lifecycle

4.2.1 Anwendungsinitialisierung

Nach der ersten Installation der Anwendung muss diese mit den Primärdaten initialisiert werden.

Dabei werden die Benutzerdaten der lizenzierten Mitglieder für das laufende Jahr direkt über die Datenbank durch den IT-Verantwortlichen in das System geladen. Dabei wird allen Benutzern der Berechtigungscode 0 zugeteilt.

Die Benutzer der Geschäftsstelle, die Administratoren seitens des SJV, werden über die Anwendung neu erstellt und erhalten den Berechtigungscode 2.

Für das laufende Jahr werden jetzt die Benutzer für die Kursadministration über die Bedienoberfläche erstellt, diese erhalten den Berechtigungscode 1.

Während des ersten Betriebsjahres werden gemäss den Anforderungen weitere Benutzer erstellt oder gelöscht, diese erhalten gemäss ihren Zugriffsansprüchen entsprechend den Berechtigungscode 0,1 oder 2.

4.2.2 Iterative Phase der Jahresnutzung

Zu Beginn einer neue Iteration der Nutzung, also eines neuen Lizenzjahres, müssen die Daten aktualisiert werden. Der IT-Verantwortliche löscht dafür zuerst alle Benutzer mit Berechtigungscode 0 aus der Datenbank. Danach folgt, wie in der Initialisierungsphase, das Laden der Benutzerdaten für das laufende Lizenzjahr.

4.3 Zielerreichung

Nach dem die Implementierung ausführlich beschrieben wurde, werden nun die Anforderungen dem Endresultat gegenübergestellt.

Zur Erinnerung folgen nochmal die funktionalen und nichtfunktionalen Anforderungen:

- Lauffähigkeit auf mobilen Endgeräten
- Keine langen Ladezeiten
- Zugangsschutz
- Kontrollierbarkeit von Lizenzen
- Login-Möglichkeit
- Funktionalität zum Erfassen von Personen
- Funktionalität zum Löschen von Personen

Die oberhalb erwähnten Anforderungen werden von der jetzigen Version der Anwendung erfüllt. Die Ladezeit ist situationsabhängig: Bei schlechter Netzwerkverbindung und bei einer grossen Anzahl Suchergebnisse ist sie möglicherweise erhöht. Allerdings ist es relativ schwierig, diese Ladezeit zu optimieren, ohne den Nutzen der Anwendung bei normaler Verbindung zu verringern.

Am meisten Verbesserungspotenzial gibt es bei der Personenerfassung. Hier wäre es eventuell hilfreich, eine Möglichkeit zur Massenverarbeitung einzurichten. Dies ist jedoch zum jetzigen Zeitpunkt nicht notwendig und könnte im Rahmen eines Change Request ergänzt werden.

Das jetzige Produkt ist voll funktionsfähig und deckt die durch den SJV gestellten Anforderungen.

4.4 Weiteres Vorgehen

Nach dieser Arbeit wird das Projekt mit grosser Sicherheit nicht zur Seite gelegt.

Wie und wann es in den Produktivbetrieb eingeführt wird, ist noch unklar. Der weitere Verlauf wird anschliessend mit dem Verband abgesprochen.

Wie bereits angesprochen, ist es auch denkbar, dass es zu einem späteren Zeitpunkt noch Erweiterungen oder Anpassungen gibt.

Denkbare Erweiterungen umfassen Mehrsprachigkeit und eine Schnittstelle, über die man CSV-Dateien importieren kann, damit man nicht direkt in die Datenbank importieren muss.

In der weiteren Zukunft besteht ein Ziel darin, dass die Datenbank automatisch durch eine REST-Schnittstelle, welche von der primären Fachanwendung zur Verfügung gestellt wird, synchronisiert wird. Mit der aktuellen Version der Verwaltungsoftware ist dies jedoch noch nicht möglich.

5 Exkurs NoSQL

5.1 Einleitung

Recht früh stellte sich die Frage, ob eine Klassische SQL-Datenbank wie MySQL oder MS Access oder eine weniger verbreitete NoSQL-Datenbank verwendet werden sollte. Im Verlauf dieses Kapitel wird zuerst NoSQL im Allgemeinen erläutert. Anschliessend wird der Dokumentenspeicher MongoDB diskutiert und zum Abschluss aufgezeigt, weshalb die Wahl auf diese Datenbanktechnologie fiel.

5.2 Geschichte und Begrifflichkeit

«Nach SQL kommt NoSQL.» Dieses Zitat der Autoren Joachim Arrasz und Christian Mennerich [AM15] trifft die Entwicklung der Datenbanken recht genau.

Als erstes muss der Begriff geklärt werden, was sich als nicht ganz einfach herausstellt.

Der Begriff NoSQL steht für «Not only SQL» und soll darauf aufmerksam machen, dass SQL-Datenbanken nicht immer die optimalen Lösungen sind. Es sollte vielmehr hinterfragt werden, was für eine Datenbank verwendet werden sollte, bevor auf ein Produkt zugegriffen wird, wie aus dem Artikel von Joachim Arrasz und Christian Mennerich zu entnehmen ist [AM15].

Eine genaue Definition von NoSQL ist fast nicht möglich. Dennoch nennen Arrasz und Mennerich eine Reihe von Punkten, welche eine NoSQL Datenbank ausmachen können:

- Die verwendeten Datenmodelle und Schemata sind nicht relational.
- Restriktionen an die Schemata sind schwach oder gar nicht vorhanden.
- Die verwendeten Technologien sind Open Source.
- Ein Hauptaugenmerk der Datenbank liegt von vornherein auf guter horizontaler Skalierbarkeit.
- Das System bringt Mechanismen zur einfachen Replikation der gespeicherten Daten mit.
- Als Konsistenzmodelle können BASE und Eventual Consistency Anwendung finden.
- Die APIs für Anfragen an das System sind einfach gehalten und unterstützen oft keine komplexen Anfragen.

Die ersten Datenbankmodelle, welche man anhand dieser Kriterien als NoSQL einordnen konnte, kamen relativ früh auf – angefangen mit DBM, einer Key/Hash-Datenbank von Ken Thomsen aus dem Jahr 1979. NoSQL als Begriff tauchte jedoch erst mit der Datenbank von Carlo Strozzi zum ersten Mal auf. Doch erst mit dem Web-2.0-Paradigma wurde NoSQL im Jahre 2000 bekannt: Durch das Verarbeiten von grossen Datenmengen wurde der Bedarf nach einer effizienteren Alternative zu relationalen Datenbanken immer grösser [vgl. Edl11, S.1].

Zum jetzigen Zeitpunkt kann man NoSQL-Datenbanken grob in vier Gruppen klassifizieren (vgl. Artikel von Joachim Arrasz und Christian Mennerich, JAXenter):

- Key-Value Stores
- Wide Column Stores
- Dokumentenorientierte Datenbanken
- Graphdatenbanken

Wenn nun SQL, NoSql gegenübergestellt wird kann man auf einige stärken und Schwächen der einzelnen Ansätze zu sprechen kommen.
An dieser Stelle werden nur drei Hauptunterschiede. Die ersten zwei genannten Punkte stammen aus der Vorlesung von Prof.Dr.Erhard Rahm [Erh11]. Es gibt noch mehr Unterschiede, die jedoch stark von der NoSQL-Unterkategorie abhängig sind.

- Skalierbarkeit
  Die Skalierung ist einer der wichtigsten Gründe für die Popularität von NoSQL. Bei SQL-Datenbanken wird die Skalierung relativ schnell sehr aufwendig. NoSQL-Datenbanken können hingegen durch die sehr geringe oder gar nicht vorhandene Struktur viel einfacher skaliert werden. Wichtig an dieser Stelle ist, dass von horizontaler Skalierung die Rede ist. Dafür ist es wichtig, dass die Aufgaben parallelisierbar sind.

- Performance
  Durch die geringen Schemarestriktionen haben NoSQL-Datenbanken im Vergleich zu SQL-Datenbanken eine grösseres Performancepotenzial. Dies manifestiert sich vor allem bei grossen Datenvolumen, wenn SQL-Datenbanken an ihren Grenzen stossen.

- Konsistenz
  Die Konsistenzgarantien sind bei den meisten NoSQL-Datenbanken im Vergleich zu SQL-Datenbanken schwächer. Im Vergleich zu SQL-Datenbanken, welche ACID-Eigenschaften besitzen, erfüllen die meisten NoSQL-Datenbanken nur BASE-Anforderungen [vgl. Edl11, S. 5].

5.3 MongoDB und Anwendung im Projekt

Für die Anwendung zur Lizenzkontrolle wird eine MongoDB-Datenbank verwendet. Dabei handelt es sich um eine schemafreie, dokumentenorientierte Datenbank.

MongoDB ist ein relativ ausgereiftes Open-Source-Projekt von dem Unternehmen 10gen aus New York (vgl., NoSQL 2011, S. 148).

Die Datenspeicherung erfolgt bei MongoDB in dem Datenformat BSON, dabei handelt es sich um ein binäres, an JSON orientiertes Format (ebd., NoSQL 2011, S. 148).

Für dies Projekt wäre mit Sicherheit auch eine MySQL-Datenbank einsetzbar gewesen, denn es handelt sich um relativ wenige, nicht komplexe Daten.

Da aber der grosse Nachteil von NoSQL-Datenbanken – dass sie nicht ACID-konsistent sind – nicht problematisch ist, wurde hier auf eine NoSQL-Datenbank gesetzt.

Dabei wurde MongoDB als optimal empfunden, da das Backend in JavaScript programmiert ist und MongoDB eine relativ grosse Entwickler- und Nutzergemeinschaft hat, durch die viele Aspekte dokumentiert sind.

6 Rückblick

Rückblickend über diese ganze Zeit kann ich nun über einige Erfolge und neuen Erkenntnisse schauen.

Ich habe festgestellt, dass das frühzeitige Erstellen eines Projektplanes die Arbeit und Koordination des Projektes erheblich vereinfacht. Wichtig ist dabei, diese Planung immer bei Bedarf anzupassen.

Das Ausarbeiten einer Lösung mit Hilfe technischer Hilfsmittel für ein gegebenes Problem hat mir viel Freude bereitet und war für mich ein interessanter Aspekt dieser Arbeit. Dass es sich um reale Nutzerbedürfnisse handelt, hat mich dabei zusätzlich motiviert.

Bei der Umsetzung meiner Vorstellung eines Endproduktes konnte ich sehr viel lernen. Dadurch, dass ich für mich neue Technologien verwendete, musste ich mir vieles von Grund auf erarbeiten, was die Arbeit umso lehrreicher und interessanter machte.

NoSQL-Datenbanken, über die ich einen kleinen theoretischen Teil dieser Arbeit geschrieben habe, waren für mich von Anfang an sehr interessant. Ich kannte diese Art von Datenbanken noch nicht und denke im Nachhinein, dass ich viel von dem Kontakt mit dieser neuen, zukunftsweisenden Technologie profitieren konnte.

A
Source Code

0

```
32      onRegisterSubmit(){
33        const user = {
34          firstName: this.firstName,
35          lastName: this.lastName,
36          email: this.email,
37          sjvNr: this.sjvNr,
38          role: this.role,
39          password: this.password,
40          lizenz: this.lizenz,
41          geburtstag: this.geburtstag
42        }
43
44      // Required Fields
45
46          if(!this.validateService.validateRoleAdmin()){
47            this._flashMessagesService.show(
48              'Sie haben nicht ausreichende Berechtigung um einen Benutzer hinzuzufügen',
49              { cssClass: 'alert-danger', timeout: 3000 }
50            );
51
52            return false;
53          }
54
55          if(!this.validateService.validateRegister(user)){
56
57            this._flashMessagesService.show(
58              'Bitte alle mit einem * markierte Felder ausfüllen',
59              { cssClass: 'alert-info', timeout: 3000 }
60            );
61
62            return false;
63          }
64
65          if((user.geburtstag != "" ) && (user.geburtstag != undefined)){
66            if(!this.validateService.validateLogicGeburtstag(user)){
67              this._flashMessagesService.show(
68                'Geburtstag bitte im Format tt-mm-jjjj oder tt.mm.jjjj. einfügen.',
69                { cssClass: 'alert-info', timeout: 10000 }
70              );
71              return false;
72            }
73          }
74
75
```

```
75
76          if(!this.validateService.validateLogicPassword(user)){
77            this._flashMessagesService.show(
78              'Person mit Zugangsrechte brauchen eine E-Mail Adresse und ein Passwort',
79              { cssClass: 'alert-info', timeout: 5000 }
80            );
81            return false;
82          }
83
84          if(!this.validateService.validateLogicLicense(user)){
85            this._flashMessagesService.show(
86              'Falls die Person lizenziert ist muss eine SJV Nr. eingetragen werden',
87              { cssClass: 'alert-info', timeout: 5000 }
88            );
89            return false;
90          }
91
92
93          this.authService.registerUser(user).subscribe(data => {
94          if(data.success){
95            this._flashMessagesService.show(
96              'Die Person wurde eingetragen',
97              { cssClass: 'alert-success', timeout: 2000 }
98            );
99
100             this.felderLehren(); // damit die felder nach dem senden gelehrt werden
101
102         } else {
103           this._flashMessagesService.show(
104             'Die Person konnte nicht hinzugefügt werden',
105             { cssClass: 'alert-danger', timeout: 3000 }
106           );
107
108         }
109       },
110       err => {
111         this._flashMessagesService.show(
112           'bitte melden sie sich zuerst an',
113           { cssClass: 'alert-danger', timeout: 3000 }
114       );
115         this.router.navigate(['login']);
116         return false;
117
118       });
119
120
121
122     }
123
```

```
  onRegisterSubmit(){
    const user = {
      firstName: this.firstName,
      lastName: this.lastName,
      email: this.email,
      sjvNr: this.sjvNr,
      role: this.role,
      password: this.password,
      lizenz: this.lizenz,
      geburtstag: this.geburtstag
    }

  // Required Fields

      if(!this.validateService.validateRoleAdmin()){
        this._flashMessagesService.show('Sie haben nicht ausreichende Berechtigung um einen Bentzer hinzuzufügen', { cssClass: 'alert-danger', timeout: 3000 });

        return false;
      }

      if(!this.validateService.validateRegister(user)){

        this._flashMessagesService.show('Bitte alle mit einem * markierte Felder ausfüllen', { cssClass: 'alert-info', timeout: 3000 });

        return false;
      }

      if((user.geburtstag != "" ) && (user.geburtstag != undefined)){
        if(!this.validateService.validateLogicGeburtstag(user)){
          this._flashMessagesService.show('Geburtstag bitte im format tt-mm-jjjj oder tt.mm.jjjj. einfügen.', { cssClass: 'alert-info', timeout: 10000 });
          return false;
        }
      }


      if(!this.validateService.validateLogicPassword(user)){
        this._flashMessagesService.show('Person mit zugangsrechte brauchen eine email adresse und ein passport', { cssClass: 'alert-info', timeout: 5000 });
        return false;
      }

      if(!this.validateService.validateLogicLicense(user)){
        this._flashMessagesService.show('Falls die person lizenziert ist muss eine sjv nr eingetragen werden', { cssClass: 'alert-info', timeout: 5000 });
        return false;
      }


      this.authService.registerUser(user).subscribe(data => {
      if(data.success){
        this._flashMessagesService.show('Die Person wurde eingetragen', { cssClass: 'alert-success', timeout: 2000 });

          this.felderLehren(); // damit dei felder nach dem senden gelert werden

      } else {
        this._flashMessagesService.show('Die Person konnte nicht hinzugefügt werden', { cssClass: 'alert-danger', timeout: 3000 });

      }
    },
    err => {
      this._flashMessagesService.show('bitte loggen sie sich zuerst ein', { cssClass: 'alert-danger', timeout: 3000 });
      this.router.navigate(['login']);
      return false;

    });



  }
```

B
Quellen

[Edl11]

Stefan Edlich, Achim Friedland, Jens Hampe, Benjamin Brauer und Markus Brückner. NoSQL: Einstieg in die Welt Nichtrelationaler Web 2.0 Datenbankern, Hanser 2011, ISBN 978-3-446-42753-2.

[AM15]

Joachim Arrasz und Christian Mennerich. Eine kleine Reise durch NoSQL: Nach SQL kommt NoSQL, JAXenter 2015. [Verwendet am 20.07.2017, https://jaxenter.de/eine-kleine-reise-durch-nosql-nach-sql-kommt-nosql-28480].

[Rah11]

Prof. Dr. Erhard Rahm, Charakteristika und Vergleich von SQL- und NoSQL-Datenbanken, Universität Leipzig 2011, [Verwendet am 20.07.2017, https://dbs.uni-leipzig.de/file/seminar_1112_tran_ausarbeitung.pdf].

C
Web Quellen

[Web1]

https://aws.amazon.com/de
zuletzt verwendet am 22.08.2017.

[Web2]

https://www.mongodb.com/
zuletzt verwendet am 22.08.2017.

[Web3]

https://nodejs.org/en/
zuletzt verwendet am 22.08.2017.

[Web4]

https://expressjs.com/
zuletzt verwendet am 22.08.2017.

[Web5]

https://angular.io/docs
zuletzt verwendet am 22.08.2017.

[Web6]

https://www.typescriptlang.org/
zuletzt verwendet am 22.08.2017.

[Web7]

http://www.putty.org/
zuletzt verwendet am 22.08.2017.

[Web8]

https://winscp.net/eng/index.php
zuletzt verwendet am 22.08.2017.

D
Verwendete Tutorials

[1] YouTube Tutorial von Learn Coding Tutorials.
https://www.youtube.com/playlist?list=PLX2HoWE32I8Nkzw2TqcifObuhgJZz8a0U
(verwendet im April 2017).

[2] YouTube Tutorial von Traversy Media.
https://www.youtube.com/playlist?list=PLillGF-RfqbZMNtaOXJQiDebNXjVapWPZ
(verwendet im Mai 2017).

E

Verwendetes Template

Autoren

Dieses Template wurde durch Pedro De Almeida auf der Basis eines Template von Patrik Fuhrer kreiert(*Software Engineering Group, University of Fribourg, Switzerland*).

Lizenz