Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelorarbeit Medieninformatik

Dezentrale Analysen mit dem Tübinger Personal Health Train (PHT) ermöglicht durch die Implementierung von einem Central User Interface (UI)

Peter Placzek

30.07.2020

Gutachter

Prof. Dr. Andreas Schilling
Medieninformatik
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Betreuer

Marius Herr
Methoden der Medizininformatik
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Placzek, Peter:
Dezentrale Analysen mit dem Tübinger Personal Health Train (PHT) ermöglicht durch die Implementierung von einem Central User Interface (UI)
Bachelorarbeit Medieninformatik
Eberhard Karls Universität Tübingen
Martikelnummer: 4062167
E-Mail: peter.placzek@student.uni-tuebingen.de
Bearbeitungszeitraum: 1.04.2020-30.07.2020

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Implementierung eines Prototypen des Central User Interfaces (UI) für den Tübinger Personal Health Train (PHT). Das UI soll es dabei ermöglichen, Dienste des PHTs zu steuern.
Dabei soll ein Nutzer nach erfolgreicher Authentifzierung im UI einen Antrag erstellen können.
Dieser Antrag ist Voraussetzung um einen sogenannten Analyse-Zug zu erstellen. Innerhalb dieses Antrages können Informationen angegeben werden, wie z.b für welche Stationen bzw. Krankenhäuser der Antrag ausgestellt werden soll.
Nachdem ein Antrag erstellt und von den betroffenen Stationen genehmigt wird, kann ein sogenannter Zug erstellt und anschließend verwaltet werden.
Desweiteren existiert ein Administrationsinterface über das unteranderem Benutzer & Berchtigungen verwaltet werden können.
Um die Bedienung als auch die Benutzerfahrung so gut wie möglich einschätzen und dann auch verbessern zu können, wurde eine Nutzerevaluation zwischen zwei Nutzergruppen durchgeführt. Dabei wurde versucht zu ermitteln, ob die Durchführung einzelener Schritte in einer akzeptablen Zeit möglich ist, als auch wie intuitiv die Bedienung dabei ist und wie man diese gegebenenfalls verbessern kann.

Danksagung

An der Stelle möchte mich ganz herzlich bei meinen Eltern Dipl.-Ing. Sabine Placzek & Dipl.-Ing. Stephan Placzek für ihre Unterstützung bedanken. Desweiteren danke ich Marius Herr für die Betreuung meiner Bacheloarbeit als auch Michael Graf für seine Anregungen und Empfehlungen.
Auch bei dem Team des PHT möchte ich mich für den Rahmen, in dem meine Bachelorarbeit stattfinden konnte, bedanken.
Zu guter Letzt danke ich allen Korrekturlesern und den Teilnehmern der Nutzerevaluation.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CRUD	Create-,Read-,Update-,Delete- (operations)
CSS	Cascading Style Sheets
DBMS	DataBase Management System
dev	Development
HTML	HyperText Markup Language
JS	JavasScript
JSON	JavaScript Object Notation
MPA	multi-Page appication
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
NPM	Node Package Manager
PHT	Personal Healt Train
req	Request
res	Response
SPA	Single-Page Application
SSR	Server-Side-Rendering
TCP	Transmission Control Protocol
TS	TypeScript
URL	Uniform Re-source Locator
URI	Unique-Resource-Identifier
WWW	World Wide Web
XML	Extensible Markup Language

Kapitel 1

Einleitung

1.0.1 Motivation der Arbeit

Die Motivation der Arbeit war es für den **Personal Health Train (PHT)** ein **Central User Interface** bereizustellen, über das andere Komponenten, wie z.B TrainBuilder, Vault,... des PHTs gesteuert werden sollen, um dezentrale Analysen zu ermöglichen.

1.0.2 Ziel der Arbeit

Ziel war es die Kontrolle für Aktionen mithilfe von Authentifizierung & Authorisierung, Daten-speicherung & -verwaltung, Validierung, ... zu gewährleisten & abzusichern.
Dabei sollte auch ein möglichst nutzerfreundliches, leicht zu bedienendes & performantes User Interface nach neusten Standards der Webtechnologien entstehen.
Dies sollte ferner zwischen zwei Nutzergruppen in einer Umfrage evaluiert werden.

1.0.3 Überblick der Arbeit

Die Arbeit beginnt mit dem **Hintergrund** in Abschnitt 2, in der zunächst die Idee und die wichtigsten Komponenten des PHT aus Sicht des UI beschrieben & dargestellt werden. Dann werden die Technologien, die **Allgemein** aber auch im Unterschied **Frontend** & **Backend** verwendet wurden, aufgezeigt.

In Abschnitt 3 **Methoden** wird dann die Methodik beschrieben, wie **Frontend** und **Backend** konzepiert und gestaltet wurde. Dabei wird auch auf die Realisierung im Kontext der zuvor beschrieben Technologien einge-

gangen und wie man diese sich vorstellen kann.
Der letzte Punkt der **Methoden/Methodik** beschäftigt sich mit der **Nutzerevaluation** und geht dabei auf Fragen ein wie:
Wie wurde die Nuzterevaluation gestaltet? Welche Aufgaben wurden gestellt? Welche Nutzergruppen haben Teilgenommen?
Die daraus entstande Nutzerevaluation wird dann in Abschnitt 4 ausgewertet und in Abschnitt 5 anschließend disktutiert.

Der letzte Abschnitt 6 dient lediglich als Anlage für Beispiele die innerhalb der anderen Abschnitte referenziert werden.

Kapitel 2

Hintegrund

2.1 PHT

2.1.1 Idee des PHT

Da Machine Learning im klinischen Kontext eine immer wichtigere Rolle spielt und man die daraus resultierenden Prognosen auf möglichst vielen Daten stützen möchte, ist es eine wichtige Frage wie man auf Patientendaten auch hinsichtlich Datenschutz an meheren Standorten Analysen betreiben kann.

Aus Datenschutzgründen und auch hinsichtlich der unterschiedlichen Primärsysteme vor Ort können Daten nicht ohne weiteres versendet werden.
Daher gibt es viele Ansätze wie man verteilte Analysen ermöglichen kann, ohne das die Patientendaten das Krankehaus verlassen und einem Risiko ausgesetzt werden.
Ein Vorschlag dafür ist der Personal Health Train (PHT), an dem jeder Teilnehmen kann, der ein Manifest [sitf] unterzeichnet.
Die Idee des PHT ist es, einen sogenannten Zug mit einem Algorithmus und weiteren Informationen zu beladen, der dann an einer Station hält die ein Krankenhaus repräsentiert. Der Algorithmus innerhalb des Zuges wird auf den dortigen Daten ausgeführt. Iterativ wandern dann die verschlüsselten Ergebnisse von Station zu Station, bis der Zug terminiert und dem Nutzer aggregierte Ergebnisse zum Download bereitstellt.

2.1.2 Dienste des PHT

Die Dienste die in der Grafik in Abbildung 2.1 dargestellt sind, werden kurz erklärt, um ein besseres Verständnis davon zu vermitteln und zu bekommen.

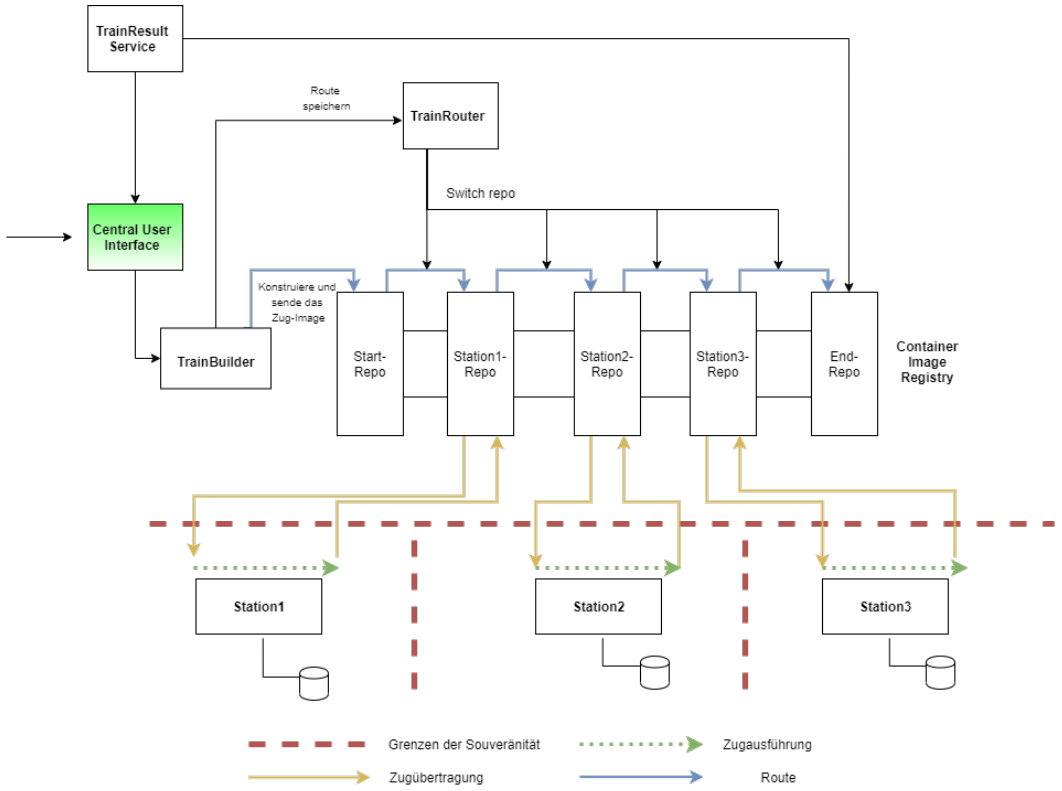


Abbildung 2.1: Übersicht der Dienste des PHT - Ursprüngliche Grafik von [Zim]

Container Image Registry

Die **Container Image Registry** umfasst sämtliche Verzeichnisse, die für die Ausführung eines Zuges benötigt werden und wird momentan durch Harbor [sitb] bereitgestellt.
Zu diesen Verzeichnissen gehört das Startverzeichniss ("Start-Repo"), finale Zielverzeichniss ("End-Repo") und die eingehenden Verzeichnisse der Stationen ("Station {1,2,3}-Repo), über die Sie einen Zug erhalten können.

Station

Verwaltet die Ausführung eines DockerImages, welches eine Station repräsentiert und lädt dieses mit den Resultaten, die dabei entstanden sind, zurück in die **Container Image Registry**.

TrainBuilder

Der **TrainBuilder** wird dazu verwendet, um einen sogenannten "Zug", der im **Central-User-Interface** konfiguriert wird, zu erstellen.

Der daraus resultierende "Zug" wird auch als "TrainImage" bezeichnet. Dieses wird in das Startverzeichnis der **Container Image Registry** überführt, das noch keiner Station zugeordnet ist. Zudem wird die Route, die der Zug gehen soll, an den **TrainRouter** übermittelt.

TrainRouter

Der **TrainRouter** überblickt die Route von jedem Zug, über die er in Kenntnis gesetzt wurde.
Dabei überprüft er für jeden Zug, wo dieser sich befindet und in welches Verzeichniss der **Container Image Registry** der Zug als nächstes übertragen werden soll.

OfflineTool

Mithilfe des **OfflineTool**s, kann ein **öffentlicher** und **privater Sicherheitsschlüssel** generiert werden.
Der **öffentliche Sicherheitsschlüssel** muss dabei vor Beginn der Konfiguration eines Zuges im **Central-User-Interface** hochgeladen werden. Dieser wird benötigt, um mithilfe des **TrainBuilder** einen Hash über die Konfiguration zu erstellen.
Dieser muss dann unter Verwendung des **privaten Sicherheitsschlüssels** im **OfflineTool** signiert werden.

TrainResult Service

Der **TrainResult Service** dient dazu, das Central-User-Interface über terminierte bzw. beendete Züge zu informieren. Desweiteren soll dann in dem UI die Ergebnisse für den Benutzer zu Verfügung gestellt werden.

2.2 Technologien

2.2.1 Allgemein

Sprachen

Bei allen Applikationen des Central User Interfaces (UI) wurde **JavaScript (JS)** oder ein Superset, wie zum Beispiel **TypeScript (TS)** verwendet.

JS wurde *"ursprünglich 1995 von Netscape für dynamisches **HTML** in Webbrowsern entwickelt [...], um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von*

***HTML** und **Cascading Style Sheets (CSS)** zu erweitern"* [sitc].
Seit geraumer Zeit hat es aber auch in Open-Source-JavaScript-Laufzeitumgebungen wie Node.js, die unabhängig vom Browser agieren, einen Platz gefunden.
Der Sprachkern von **JS** wird durch den sogenannte ECMA Script standartisiert und muss oft je nach Laufzeitumgebung indivivudell realisiert werden.

Interpreter

Als Interpreter wurde sowohl für die Frontend- als auch Backend-Dienste **Node.js** verwendet. Node.js ist eine "Open-Source-JavaScript-Laufzeitumgebung, die JavaScript-Code außerhalb eines Webbrowsers ausführen kann" [site], die auch als Webserver mithilfe eines serverseitigen Open-source Webframework wie express.js fungieren kann.

"Node.js enthält einige Module, die direkt in das Binärpaket kompiliert wurden. Dazu gehören neben dem Modul für asynchronen Netzwerkzugriff auch Adapter für das Dateisystem, Puffer, Zeitgeber und eine allgemein gehaltene Datenstrom-Klasse" [site].

Architektur

Das generelle Ziel bei der Konzepierung des Cental-UI's war es, eine klare Trennung & geringe Kopplung von **Frontend** (Darstellung) und **Backend** (Verwaltung) zu erreichen.
Die größten Vorteile bei diesem Vorgehen sind unteranderem:

- Der Server muss nicht mehr die vollständige Seite (z.B **HyperText Markup Language (HTML)** für jeden Nutzer generieren und bereitstellen, sondern kann über eine API Schnittstelle die erfoderlichen Daten in einem frei wählbarem Format (z.B **JavaScript Object Notation (JSON)**, **Extensible Markup Language (XML)**) der Frontend-Applikation anbieten.

- Frontend und Backend können dezentral auf verschiedenen Server laufen. Zudem kann man beides individuell mit Technologien wie z.B Load-Balancing skalieren.

- Hohe Performance

2.2.2 Frontend

Die Darstellung wurde dabei als **Singe-Page-Application (SPA)** realisiert, um so gegenüber einer herkömmlichen Webanwendung nicht zwischen HTML-Dokumenten zu verlinken, sondern die Inhalte innerhalb eines HTML-Dokuments dynamisch zu verändern.

Sprachen

Bei der Frontendapplikation wurde sowowhl **HTML**, **CSS** und, wie bereits beschrieben, die Skriptsprache **JS** der **World Wide Web (WWW)** Sprachen verwendet.

HTML ist eine Auszeichnungssprache und dient der Strukturierung bzw. Anordnung von Bildern, Videos, Texten, Links und vielem mehr in Form von sogenannten HTML-Elementen innerhalb einer Website.
Diese HTML-Elemente haben auch als Eigenschaft, dass sie immer aus einem OpenTag und in den meisten Fällen auch aus einem CloseTag bestehen. Der CloseTag soll dabei das Ende eines Elements symbolisieren. HTML-Elemente werden daher oft durch den Inhalt zwischen den beiden Tags beschrieben. Zuden kann aber der OpenTag mit Attributen versehen werden, der das Element mit zusätzlichen Informationen bzw Eigenschaften beschreibt. Diese können zum Beispiel CSS Eigenschaften sein, die nicht über eine sepearte Datei oder HTML-Style-Element definiert wurden.

Die Formatierungssprache **CSS** dient, wie auch schon angedeutet, dabei Gestaltungsanweisungen für HTML Elemente festzulegen. Dabei können diese Anweisungen direkt an ein HTML-Element über ein HTML-Style-Attribut gebunden werden.
Die andere und deutlich elegantere Variante ist, die Zuweisung über ein seperate Datei oder ein HTML-Style-Elemente mit sogenannten Klassen zu realisieren. Dabei werden für diese Klassen sogenannte Blöcke definiert, die aus meheren Anweisungen bestehen können. Da eine Klasse auch mehreren HTML-Elementen als Attribut zugwiesen werden kann, kann man so oftmals unnötige Redundanz vermeiden.

Framework - Vue.js

Als Framework wurde nach Vorgabe das Javascript-Webframework Vue.js verwendet. Dieses dient dem Erstellen von **SPA's** nach dem **Model-View-ViewModel (MVVM)** Schema. Es findet aber auch Anwendung in bereits bestehenden **Multi-Page-Applikationen (MPA's)**, in denen

das Framework lediglich in einem fest definierten Bereich genutzt werden kann.

Das **MVVM** Muster ist eine Variante des auch in vielen anderen Sprachen auffindlichen und bekannten **Model-View-Controller (MVC)** Prinzips. Bei dem **MVC** Muster fungiert der Controller als zentrales Element für die Steuerung von View und Modell. Der Controller wird dabei durch Benutzerinterkationen des Views darüber in Kenntnis gesetzt, Anpassungen an dem View und auch gegebenenfalls an denen im Modell befindlichen Daten vorzunehmen.
Das **MVVM** hingegen verzichtet bei seinem Ansatz auf den Controller und realisiert die notwendigen Anpassungen, die möglicherweise durch Benutzerinteraktionen erfoderlich sind, durch das ViewModel. Dieses bindet Ereignisse aus dem UserInterface (View) an eigene Seteuerelemente und bindet die in der Folge von dem Model manipulierten Daten wieder an den View.
Die wichtigsten Elemente von Vue.js, die innerhalb der Applikation verwendet werden, sind **Komponenten** und der **Store**.

Framework - Vue.js Komponenten

Wie bereits erwähnt sind Komponenten zentrale Elemente einer **SPA**/Vue.js [Youb] Applikation.
In ihrer Grundform sind sie *"wiederverwendbare Vue.js Instanzen, die über einen eindeutingen Namen identifiziert werden"* [Pet00]. Eine solche Komponente könnte zum Beispiel wie folgt aussehen und definiert werden.

```
<script>
    Vue.component('button-counter', {
      data: function () {
        return {
          count: 0
        }
      },
      template: '<button v-on:click="count++">' +
                    '{{ count }}' +
                '</button>'
    })
</script>
```

Codeausschnitt 2.1: Button Counter - [Pet00]

Mit diesem Syntax hat man eine unlimitierte wiederverwendbare Komponente unter dem Alias und auch **HTML**-Tag **button-counter**, wie in Abbildung 2.1 definiert ist.

Das bedeutet, wenn ein **HTML** Element mit diesem Alias versehen wird und sich innerhalb einer Vue.js **Haupt-Komponente** befindet, wie in Codeausschnitt 2.2 zu sehen, wird diese nach obriger Definition dargestellt.

```
<script>
    new Vue({ el: '#root' })
</script>

<div id="app">
  <button-counter></button-counter>
</div>
```

Codeausschnitt 2.2: Haupt-Komponente - [Pet00]

Die Zentralen Bestandteile einer Komponente sind, wie oben in dem Ausschnitt 2.1 bereits ersichtlich, das **data** und **template** Objekt Attribut.
Das innerhalb des **data** Objekt Attributes zurückgegebene Objekt repräsentiert **Key-Value** Paare, die innerhalb von anderen Elementen, wie **methods**, **computed**, etc einer Vue.js Komponente manipuliert werden können, und in sogenannte **LifeCycles** [Bem] neu in das **template** gerendert werden.
Zudem können diese Werte auch direkt im View manipuliert werden.
Dieses Prinzip nennt man in bidirektionaler Form **two-way-binding** [Lud] und in direktionaler Form **one-way-binding** [Lud] und ist Teil fast jeder **SPA**.
In dem Kontext der Applikation repräsentieren sie auch einzelne Seiten.

Framework - Vue.js Store

Bei dem **Store** handelt es sich um ein **State-Management-Pattern** [Youa], das in dem Fall von der Vue.js Applikation mithilfe der Bibliothek **vuex** implementiert wurde.
Das Prinzip soll in der nachfolgenden Abbildung 2.2 verdeutlicht werden.

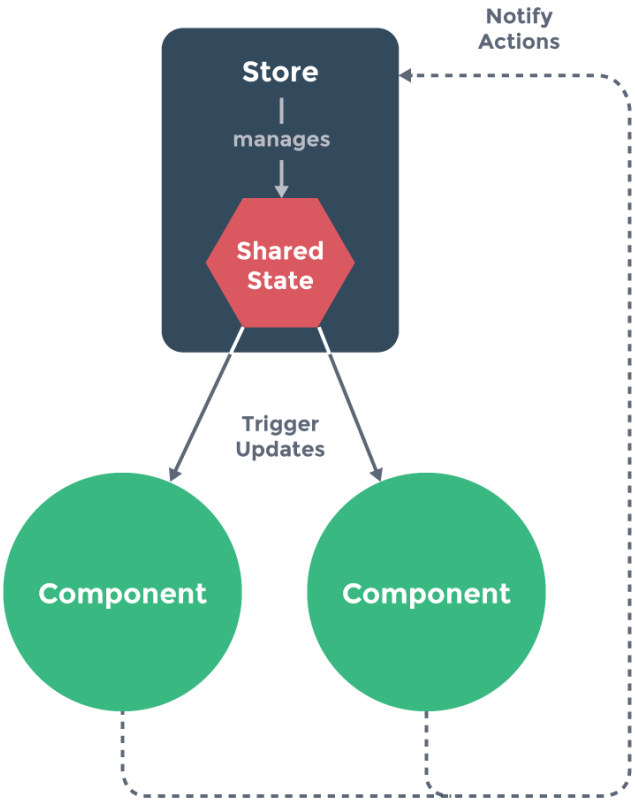


Abbildung 2.2: Abbildung "one way data flow" - Schema des **StateManagementPattern** [Youa]

Das Ziel dabei ist es, den Zustand (**State**) von Informationen (Skalare, Objekte, usw.) zu speichern und bei einer Mutation (**Mutation**) zu aktualiseren.
Dieser Store könnte wie zum Beispiel in der nachfolgenden Abbildung 2.3 minimal definiert werden, um den Wert des Zustandes **count** mithilfe der Mutation **increment** zu erhöhen.

```
const store = new Vuex.Store({
    state: {
        count: 0
    },
    mutations: {
        increment (state) {
            state.count++
        }
    },
    ...
})
```

Codeausschnitt 2.3: Beispiel eines minimalen Vuex.Store

Solche **Mutationen** können auch durch vordefinierte Aktionen ("**Actions**") eines Benutzers innerhalb einer Komponente ausgelöst werden.
Diese **Aktionen** 6.6 erhalten als erstes Argument den Store Context, der einige Methoden und Konstanten des Stores zu Verfügung stellt. Dieser Context enthält zum Beispiel die Methoden **commit**, mit der eine Mutation ausgelöst werden kann, als auch die Methode **dispatch** mit der eine andere Aktion innerhalb des Stores ausgeführt werden kann.

Desweiteren ermöglicht der Store allen Komponenten über vordefinierte **Getter** bei Veränderungen eines Zustandes informiert zu werden und die jeweiligen Informationen zu aktualisieren.
Diese Erweiterung könnte nach obrigen Beispiel in 2.3, wie in dem nachfolgenden Codeausschnitt 2.4 aussehen.

```
const store = new Vuex.Store({
    ...
    actions: {
        doIncrement({commit}) {
            commit('increment');
        }
    },
    getters: {
```

```
9           count: (state) => {
10              return state.count;
11          }
12      }
13  })
```

Codeausschnitt 2.4: Beispiel eines Vuex.Store

Hierbei wird der Store um einen gleichnamigen Getter **count** wie der Zustand erweitert, über den in einer Komponente auf den Zustand zugegriffen werden kann. Mithilfe der **doIncrement** Aktion kann von einer Komponente aus die Mutation ausgelöst werden.

Libraries - Nuxt.js

Als zentrale Library wurde Nuxt.js verwendet, welche dazu dient, universelle Applikationen schneller zu bauen. Da Vue.js relativ viel Spielraum lässt, wie man das Grundgerüst seiner Applikation gestalten kann, versucht Nuxt.js dafür einen besseren Rahmen zu schaffen und ermöglicht es unkompliziert Code und Resourcen zwischen Klient- und Server-Seite zu schicken. Zudem wird eine oftmals schon ausreichende Ordnerstruktur beim Initialisieren erstellt.

Somit kann man oftmals ganz leicht zum Beispiel zwischen **SPA** und **Server-Side-Rendering (SSR)** wechseln, um so auch bei initaler Browser Anfrage eine vom Server gerenderte Seite bereitzustellen, auf der dann durch Benutzerinterkationen nur noch einzelne Komponenten ausgetauscht werden.

Libraries - Nuxt.js Plugins

Plugins werden zum Beispiel dazu verwendet, eigene oder andere **NPM**-Module in den Applikationscontext der **Vue.js** Applikation zu injezieren.
Das Ziel ist es, ein Objekt, Funktion, Komponente, Konstante, etc. für alle Komponenten zu Verfügung zu stellen. Mithilfe der ”export default” Methode wird eine Funktion exportiert, welche die aktuelle Datei repräsentiert.
Diese Funktion kann beim Einbinden von einem anderen Modul übergeben werden.
In dem Fall wird als erstes Argument der Kontext von Nuxt.js (Vue.js Library) automatisch übergeben, der Zugriff auf interne Module und Funktionen ermöglicht. Als zweites Argument wird eine Callback Funktion zu Verfügung gestellt, mit deren Hilfe ein Skalar, Object, etc. unter einem Alias zum Beispiel **dummyAlias** 2.5 global für die Komponenten zu Verfügung gestellt wird.

```
...
export default ({ app, store, env }, inject) => {
    let dummyValue = 'abc';

    inject('dummyAlias',dummyValue);
    app.$dummyAlias = dummyValue;
}
...
```

Codeausschnitt 2.5: Plugin Beispiel

2.2.3 Backend

Sprachen

Bei beiden Backend-Applikationen wurde das SuperSet **TS** der Skriptsprache **JS** verwendet.

Frameworks - Express.js

Sowohl für den Auth-Server als auch für den Resourcen-Server wird das Webframework Express.js verwendet.
Bei diesem Framework handelt es sich, um genauer zu sein, um einen Webserver, der an einen bestimmten **Transmission Control Protocol (TCP)** Port gebunden wird und Webanfragen darüber bearbeitet.
Dabei werden für Kombinationen von **HTTP** Methoden und **Uniform Resource Locators (URLs)** sogenannte Callback-Funktionen angegeben bzw definiert. Diese Callback-Funktionen erhalten beim Aufruf zwei Parameter. Der erste Parameter beschreibt den Request, dabei kann unteranderem Request Payload ausgelesen werden. Der zweite Parameter beschreibt den Response, und besitzt Methoden mit denen der Status-Code, ContentType, etc. definiert werden können.

Frameworks - Express.js Router

Die Routing Funktionalität wird mithilfe der Library **Express.js** erzielt. Dabei kann jeweils für eine **URI** mehere Callback Funktionen je nach **HTTP**-Methode definiert werden.

Dies könnte für eine **URI** wie z.B "dummyURI" und die **HTTP Methoden** "GET" & "POST" wie in 2.6 aussehen.

```
import { Router } from 'express';

let router: any = Router();

let dummyURI: string = 'dummy-uri';

router.get(dummyURI, [], (req: any, res: any) => {
    res.send('Get Request on URI: '+ dummyURI);
});

router.post(dummyURI, [], (req: any, res: any) => {
    res.send('Post Request on URI: '+ dummyURI);
});
```

Codeausschnitt 2.6: Router Beispiel

Wie sich an dem Syntax der Methoden, die **HTTP**-Methoden repräsentieren, erkennen lässt, können **drei Parameter** übergeben werden.
Der erste gibt dabei die **URI** an, bei dem zweiten Parameter können sogenannte **Middlewares** definiert werden, die vor dem dritten Argument, welche eine **Callback Funktion** mit den Argumenten **Request (req)** & **Response (res)** ist, aufgerufen wird.
Nach obrigen Beispiel würde bei einer GET-Anfrage durch den Browser an das Backend der Text "Get Request on **URI**: dummy-uri" zurückgegeben und im Fall einer POST-Anfrage der Text "POST Request on **URI**: dummy-uri" angezeigt werden.

Eine Middleware hat ebenso auf das **req** & **res**-Objekt Zugriff.

Frameworks - Express.js Router Middleware

Mithilfe einer Middleware kann sowohl das **req**- als auch das **res**-Objekt manipuliert werden, als auch um Funktionalitäten erweitert werden.
Dabei werden sie durch das erste und zweite Argument einer Middleware Funktion repräsentiert.
Als drittes Argument erhält es jedoch eine Callback Funktion mit der die nächste Middelware im *"Anforderung/Antwort-Zyklus der Anwendung"* [sitd] augerufen werden kann, sofern die aktuelle diesen nicht unterbrochen hat.
Eine triviale Middleware könnte wie folgt realisiert werden 2.7:

```
...

let dummyMiddleware = (req: any, res: any, next: any) => {
```

```
4     console.log('dummy');
5     next();
6 };
7
8 ...
9
10 router.get(dummyURI, [dummyMiddleware], (req: any, res: any) => {
11     res.send('Get Request on URI: '+ dummyURI);
12 });
13 ...
```

Codeausschnitt 2.7: Router - Middleware Beispiel

Somit würde jedes Mal, wenn die **URI** bzw. Wert der Variable **dummyURI** aufgerufen wird, der Wert "dummy" in der Konsole dargestellt werden. Eine Middleware kann auch global und nicht nur explizit für eine Route und Methode definiert werden, sondern kann auch global für jede Route mit dem folgendem Syntax, definiert werden 2.8.

```
1 ...
2
3 router.use(dummyMiddleware);
4
5 ...
```

Codeausschnitt 2.8: Router - Middleware benutzen

Datenbank

Als Datenbank Library wird Knex.js verwendet, welche als SQL Query Buidler für verschiedene **Datenbank-Management-Systeme (DBMS)** wie MySQL, SQLite, MSSQL, ... genutzt werden kann.
Ferner können verschieden Enviroments (z.B testing, development, ...) definiert werden, die dabei verschiedene Konfiguration, auch hinsichtlich **DBMS** haben können.
Somit kann eine einfache InFile-Datenbank wie SQLite für Testen & Entwicklung genutzt werden und eine skalierbareres und perfomanteres **DBMS** für den Produktionsbetrieb.
Desweiteren können sogenannte "Migrations" und "Seeds" erstellt werden.
Bei Migrations handelt es sich um Tabellen Schema Definitionen, die sowohl Änderungen als auch Neuerungen zu der vorherigen erstellen Migration haben können. Daher handelt es sich dabei um eine ideale Lösung für die Backend Applikationen des UI.

Kapitel 3

Methoden

3.1 Fontend - Webapplikation

3.1.1 Vorwort

Die Frontend Web Applikation ist eine der drei auf Node.Js [Fou] basierenden Applikationen, die zusammen das **Central User Interface** bilden.
Dabei ist sie für das Routing und die Darstellung im Browser verantwortlich. Da es sich hierbei um eine **SPA** handelt, werden immer nur einzelne Komponenten bei einem Seitenwechsel ausgetauscht. Es werden dabei lediglich Informationen über eine **Anwendungsprogrammierschnittstelle (API)** Schnittstelle des Auth- oder Resource-Server übertragen.
Dies hat zu Folge, dass erstmaliges Seitenladen oftmals etwas mehr Zeit beansprucht, da dabei alle Abhängigkeiten (z.B Bilder, **CSS**, **HTML**, ...) geladen werden müssen, die dann auch bei nachfolgenden Anfragen aus dem Cache geladen werden können.
Ein weiterer Vorteil neben der höheren Performance ist, dass eine geringe Koplung zwischen Frontend & Backend und auch innerhalb ihrer Unterteilung erreicht wurde und der Server nicht gezwungen wird, den kompletten Seitenaufbau für den Client (z.b Webbrowser) zu generieren, sondern diese Last von dem Browser übernommen wird.

3.1.2 Installation

Wie bei allen Node.Js Applikationen, können Skriptdateien mit dem **Node Package Manager (NPM)** [Fou] gestartet werden. Dabei werden sowohl Shortcuts für Start-Befehle, Abhängigkeiten, **Entwicklungs (dev)**-Abhängigkeiten und andere Informationen in einer Konfigurationsdatei 6.1 mit dem Titel **package.json** definiert.

Beim Betrachten erkennt man schnell, dass es sich bei den Attributen **name, version, description, author & private** um sogenante Metatags handelt, die das Packet bzw. die Applikation beschreibt.

Das Objekt hinter dem Schlüssel **scripts** ist eine Möglichkeit häufig benutze Befehle mit sogenannten Aliasse zu versehen.
In diesem Fall dienen die Befehle:

- **dev** zum Starten der Applikation im Development Modus und somit auch einer in Memory Build Variante.
- **build** um die Applikation mithilfe von Webpack zu erstellen und danach mit einem Webserver (z.B Nginx oder Apache) bereitzustellen.

Die Schlüssel **dependencies** und **devDepencies** dienen dabei Abhängigkeiten zu definieren, die zum einen für die Konstruktion ("Build") der Ursprungsdateien der Applikation und zum anderen um die Applikation weiter zu entwickeln angeben, welche Packete installiert oder aktualisiert werden sollen.

3.1.3 Konfiguration

Die Konfiguration der Applikation ist relativ generisch gehalten, dabei können in einer **.env** Datei die **API URLs** des **Resource-** & **Auth-server** geändert bzw. angegeben werden.
Initial sieht die Repräsentation wie folgt aus:

```
resourceApiUrl=http://localhost:3001
authApiUrl=http://localhost:3002
```

3.1.4 Übersicht

Die nachfolgende Grafik in Abbildung 3.1, soll eine Vorstellung verschaffen, wie die Komponenten im Zusammenhang stehen.

Die Interaktion beginnt mit der **Benutzer Anfrage**, die durch den Browser des Nutzers an die Applikation gestellt wird.
Der **Router** bestimmt dann welche **Seiten-Komponente** angesprochen und dargestellt werden soll. Dabei können zusätzliche **Layout-Komponenten** eingebunden werden, bei denen es sich auch um Vue.js-Komponenten handelt.
Die **Seiten-Komponenten** können dann beliebige Services anbinden und mit dem **Store** mithilfe von Getters bestimmte Zustände abonieren oder über ein Aktion Mutationen von Zuständen herbeizurufen.

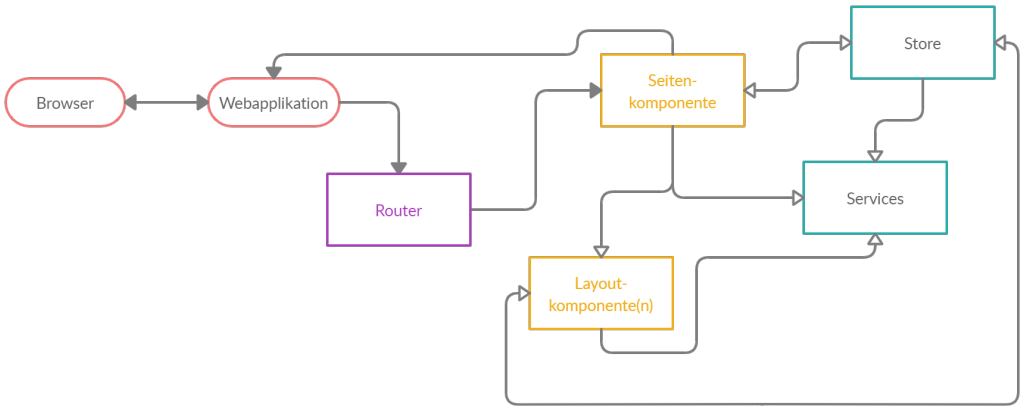


Abbildung 3.1: Interaktionsübersicht der zentralsten Elemente

3.1.5 Router

Der Router ist ein zentrales Element der Frontend Applikation. Er bestimmt welche Komponenten bei welchem **Unique-Resource-Identifier (URI)** geladen bzw. generiert werden.
Hierbei muss keine Konfiguration explizit angegeben werden, das **URL** Schema bzw. die Endpunkte werden durch die Verzeichnis-Struktur und durch Muster in den Dateinamen generiert, diese liegen dabei in dem Stammverzeichnis unter **pages**.

Middleware

Der Mechanismus einer Middleware ermöglicht es vor Routenaufruf Restriktionen oder Manipulationen des Zugriffs vorzunehmen. Diesen Mechanismus wurde für zwei zentrale Aufgaben verwendet:

Aufgabe 1: Authentifizierung & Authorisierung Hierbei sollen bestimmte Endpunkte abhängig von dem **Login-Status** & den **Berechtigungen** eines Benutzers restriktiert werden. Dabei wird dem Meta Tag einer Vue.js Komponente, die einen Endpunkt repräsentiert, Attribute definiert, mit deren Hilfe überprüft werden kann, ob ein Benutzer angemeldet ist oder ob er überhaupt über die entsprechende(n) Berechtigung(en) verfügt.
Dies sieht zum Beispiel wie im folgenden Codeausschnitt 3.1 aus.

```
1 export default {
2     meta: {
```

```
        requireLoggedIn: true,
        requireAbility: (can) => {
            return can('add','user');
        },
        ...
    },
    ...
}
```

Codeausschnitt 3.1: Meta-Attributdefinition - **Login-Status** & den **Berechtigungen**

Es können dabei drei verschiedene Attribute gesetzt werden:
Die Attribute **requireLoggedIn** & **requireGuestState** sollen dabei festlegen, ob ein Benutzer eingeloggt oder ausgeloggt sein muss oder nicht, um eine Route aufzurufen.
Mithilfe des Attributes **requireAbility**, kann eine Callback Funktion angegeben werden, die als Argument eine Funktion erhält, mit deren Hilfe man überprüfen kann, ob ein User über eine Berechtigung verfügt, beziehungsweise in welchem Ausmaß. Die Callback Funktion muss dabei einen boolischen Wert zurückgeben.

Aufgabe 2: Layout Die Aufgabe, die hierbei realisiert werden sollte war, abhängig von dem ausgewählten Menuelement (**Home** oder **Admin**) eine unterschiedliche Sidebar auf der Seite im Browser darzustellen.
Ähnlich wie bei der Authentifizierung & Authorisierung sollte hier bei jeder Vue.js Komponente die einen **URL**-Endpunkt repräsentiert, das Attribut **navigationId**, wie in in dem Beispiel des Codeausschnittes in 3.2 zu sehen, im Meta Attribute des exportierten **JS** Objektes definiert werden, welches angibt zu welchem Navigationselement es gehört.
Dabei muss vor jeder Änderung der **URL** überprüft werden, ob die momentan dargestellte Sidebar auch zu dem Navigationselement passt.
Je nach Situation muss es eine andere Sidebar abhängig von den Attributen **requireLoggedin**, **requireGuestState** & **requireAbility** die Komponente darstellen oder nicht.

```
export default {
    meta: {
        ...,
        navigationId: 'admin' // 'admin' oder 'default'
    },
    ...
```

```
7 }
```

Codeausschnitt 3.2: Meta-Attributdefinition - **navigationId**

3.1.6 Services

API

Die zentrale Komponente des **API Service** ist die **API Factory**. Diese erstellt eine Axios Instanz "Axios ist ein Versprechen basierter **HTTP** Client für Node.js und Browser" [sita].
Diese Instanz wird als Argument für eine Funktion übergeben, welche die Axios Bibliothek mit entsprechenden Decorator [Hau] Methoden und erweiterten Funktionalitäten erweitert und zurückgibt.

Grund dafür ist zum einen, dass die Bibliothek einfach ersetzt werden kann, da keine direkte Kopplung zu anderen Komponenten existiert.
Zum anderen können Konfigurationen wie zum Beispiel BaseUrl und sogenannte Interceptors für jede Instanz der **API Factory** gesetzt werden.

Mithilfe solcher Interceptors kann zum Beispiel die Antwort einer API Anfrage, die einen **ausgelaufen Authentifizierungstoken** signalisiert, der zuvor je nach Login Status des Besuchers gesetzt wurde, wieder entfert und den Benutzer auf die Login Seite weiterleitet.
Mithilfe der **API Factory**, werden dabei folgende Instanzen erstellt:

- Die **ResourceAPI** dient zur Kommunikation mit dem Backend Resource Server. Diese Instanz wird dann von einem EdgeService, der verschiedene **Create-,Read-,Update-,Delete-Operationen (CRUD)** für eine Domaine (z.B Anträge, Züge, Master Images, Stationen usw.) bereitstellt, verwendet. Mit "Edge" ist dabei die "Kante" bzw. Kommunikation zu einem API Server gemeint.

- Die **AuthAPI** verwendet die Instanz nach gleichem Schema wie die **Resource API**, jedoch wird diese von EdgeServices verwendet, die Domainen wie z.B Benutzer, Berechtigungen, Gruppen, usw. repräsentieren.

Ein Ausschnitt, des soeben beschriebenen Verfahren, ist in Abschnitt 6.2 beispielhaft dargestellt.

Edge

Die **Edge Services** umfassen jeweils eine Sammlung von **CRUD**-Operationen für eine Domaine.
Diese Edge Services haben neben neben dem Vorteil der Abstraktion den Nutzen, dass innerhalb der Applikation und die via API bereitgestellten Daten Objekte ineinander überführt werden können. Somit wird gewährleistet, dass die Kopplung zwischen Objekten in der Web Applikation und Objekten der Backend Server möglichst gering ist.
Es wird für jeden EdgeService, wie in dem Codeausschnitt 6.3 exemplarisch dargestellt, ein **Mapping** zwischen internen Objektattributen und externen Objektattributen definiert.

Storage

Der **Storage Service**, wie in dem unteren Ausschnitt 3.3 dargestellt, ist eine relativ triviale Komponente. Sie ermöglicht es unter abstrakten Methoden Daten in dem Browser zu speichern. Momentan wird für die Datenspeicherung im Browser der **LocalStorage** [sith] verwendet.

Bei dem **LocalStorage** handelt es sich um "eine Technik für Webanwendungen, mit der Daten in einem Webbrowser gespeichert werden" [sith]. Der LocalStorage ist im Vergleich zu der anderen Technik der Datenspeicherung, die sich **SessionStorage** nennt, ein persistenter Speicher.

```
1  ...
2  const StorageService = {
3      set(key,value) {
4          if (process.server) { return }
5
6          localStorage.setItem(key, value)
7      },
8      drop(key) {
9          if (process.server) { return }
10
11         localStorage.removeItem(key)
12     },
13     get(key) {
14         if(process.server) return null;
15
16         return localStorage.getItem(key);
17     },
18     ...
```

```
19 }
20 ...
```

Codeausschnitt 3.3: StorageService

Auth

Der **AuthService** besteht aus mehreren Komponenten, welche im Folgenden näher umrissen werden.

Der **AuthStorage** stellt, wie in Abschnitt 6.5 dargestellt, Methoden bereit, mit denen sitzungsbasierte Informationen wie Benutzer, Berechtigungen, etc. gesetzt, erhalten und gelöscht werden können.
Diese Informationen werden in dem **LocalStorage** mithilfe der **Storage** Komponente gespeichert.

Die gleichnamige Hauptkomponente stellt Methoden bereit, die zum Beispiel den **Authentifizierungstoken** der beiden **APIs** (Resource- & Auth-**API**) je nach internen Aktion des User Interfaces setzt oder wieder entfernt, wie dem in Abschnitt 6.4 beigefügten Codeausschnittes zu entnehmen.

Desweiteren stellt sie Methoden für Login, Logout bereit, bei denen zum Beispiel mithilfe der **AuthStorage** Komponente Informationen wie Benutzer, Berechtigungen im Browser gespeichert werden.

3.1.7 Store

Der Store findet seine Anwendung in der Applikation für die **Authenfizierung** und das **Layout**.
Im Falle der **Authentifizierung** repräsentieren die **Zustände** den aktuellen **Benutzer**, seine **Berechtigungen** und den möglichen **Token**, wie im folgenden Codeuasschnitt 3.4 zu sehen.

```
1 ...
2 const state = () => ({
3     ...
4     user: AuthStorage.getUser(),
5
6     permissions: AuthStorage.getPermissions(),
7     abilities: AuthStorage.getAbilities(),
```

```
8
9       token: AuthStorage.getToken()
10  });
11  ...
```

Codeausschnitt 3.4: AuthStore - States

Die Werte dieser Zustände werden mit dem **AuthStorage** des bereits beschriebenen **AuthServices** initalisiert.
Diese Zustände werden über gleichnamige Getter den Komponenten zum Abonnieren, wie in folgendem Codeauszugs 3.5 dargestellt, zu Verfügung gestellt.

```
1   ...
2   const getters = {
3       user: (state) => {
4           return state.user
5       },
6       permissions: (state) => {
7           return state.permissions
8       },
9
10      loggedIn: (state) => {
11          return !!state.token
12      },
13      ...
14  };
15  ...
```

Codeausschnitt 3.5: AuthStore - Getters

Für Aktionen, wie ein **User Login** der von mehreren Komponenten ausgelöst werden können soll, wurde hierfür eine vordefinierte Aktion, **triggerLogin**, wie in dem Codeausschnitt exemplarisch 6.6 dargestellt wird etabliert.

Innerhalb solcher **Mutationen**, die oftmals durch eine **Aktion** ausgelöst werden, werden nicht nur die Werte der Zustände manipuliert, sondern auch zum Beispiel der **AuthStorage**, in dem diese Werte zwischengespeichert werden.
Dies ist in der folgenden Codezeilen 3.6 abgebildet.

```
1   ...
2   const mutations = {
3       ...
```

```
4       setToken (state, token) {
5           state.token = token;
6           AuthStorage.setToken(token);
7       },
8       unsetToken(state) {
9           state.token = null;
10          AuthStorage.dropToken();
11      },
12      ...
13  }
14  ...
```

Codeausschnitt 3.6: AuthStore - Mutationen

3.1.8 Plugins

In dem Fall des **Auth Plugin**, welches als Codeausschnitt in den Appendix in Abschnitt 6.7 dargestellt wird, werden zum Beispiel Methoden über ein Objekt bereitgestellt, mit deren Hilfe überprüft werden kann, ob der aktuelle Benutzer eine Berechtigung besitzt und zu welchem Ausmaß.

3.2 Backend - Authserver & Resourceserver

3.2.1 Vorwort

Wie bereits zu der **Web Applikation** angemerkt, handelt es sich bei dem **Auth- & Resource-Server** wieder um Node.js basierte Applikationen.
In diesem Fall wird jedoch Node.js in Kombination mit einer "Webserver" Library mit dem Titel **Express.js** [sitd] verwendet, um **API**-Endpunkte für das Frontend bereitzustellen, die es unteranderem ermöglichen, Resourcen zu verwalten.

3.2.2 Installation

Die Installation verläuft wieder nach ähnlichem Schema statt, wie die Installation der Web Applikation.
Jedoch unterscheidet sich die package.json nicht nur hinsichtlich der Packete (depencies & devDepencies) die bentöigt werden, sondern auch hinsichtlich der Aliase, die in dem Objekt mit dem Schlüssel **scripts** hinterlegt sind.
In dem Fall des **Auth- & Resource-Server** stehen folgende zentrale Aliasse zu Verfügung.

- **dev** zum Starten der Applikation im Development Modus und somit auch einer in Memory Build Variante.
- **dev-watch** Ähnliche Funktionalität wie bei **dev**, aber zusätzlich werden noch auf live Änderungen in Dateien reagiert und die Applikation neu in Memory gebaut.
- **build** um die Applikation mithilfe von Webpack zu erstellen und danach mit einem Webserver (z.B Nginx oder Apache) bereitzustellen.
- Mit dem Alias **migration-up** kann eine Datenbank nach vordefiniertem Schema initialisiert oder aktualisiert werden.
- Mit **migration-down** hingegen werden alle Änderungen aus vorherigen Migrations zurückgenommen und die Datenbank zurückgesetzt.
- **seed-run** dient dazu die Datenbank mit vordefinierten Datensätzen zu befüllen. Dieses Verfahren wird oft für Testzwecke verwendet.

3.2.3 Konfiguration

Die Konfiguration findet auch hier wieder über eine zentrale Konfigurationsdatei mit dem Titel **.env** statt. Innerhalb dieser Datei können folgende Werte definiert werden.

Auth-Server

```
PORT=3002
JWT_KEY=<JWT KEY>
```

Der **PORT** dient dazu, zu spezifizieren, auf welchem **TCP** Port die Applikation laufen soll. Die **JWT_KEY**-Variable referenziert auf den Schlüssel, der bei der Authenfizierung zum Signieren des Tokens benötigt wird.

Resource-Server

```
PORT=3001
AUTH_API_URL=http://localhost:3002/
TRAIN_BUILDER_SOCKET_URL=http://localhost:3003/
```

Der **Port** dient wie bei dem **Auth-Server** zur Specifizierung des **TCP** Ports auf dem die Applikation laufen soll. Die **Auth_API_URL** gibt dabei den **Auth-Server** an, gegen den der übergebene Authentifizierungstoken validiert werden soll. Bei der **TRAIN_BUILDER_SOCKET_URL** handelt es sich um die URL, mit deren Hilfe mit dem **TrainBuilder** kommuniziert werden kann.

3.2.4 Übersicht

In der nachfolgende Grafik in Abbildung 3.2 soll ebenfalls eine Vorstellung verschafft werden, wie die Komponenten des jeweiligen Backendservers im Zusammenhang stehen.

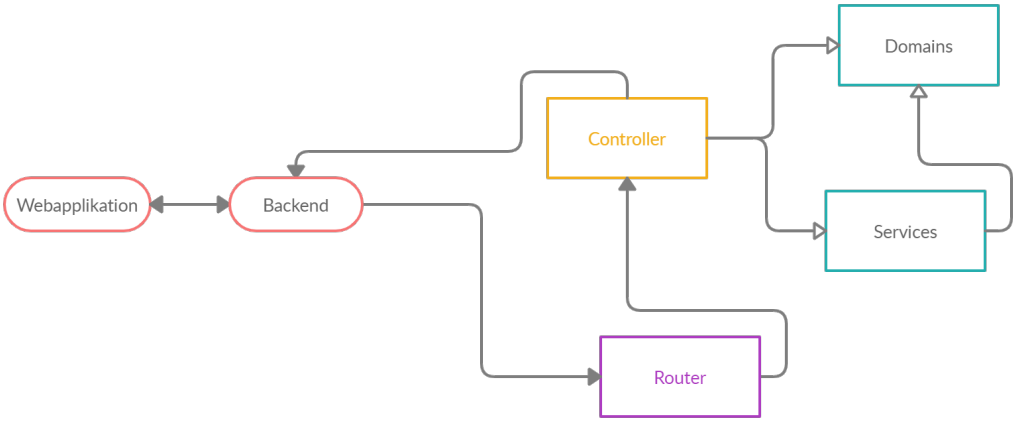


Abbildung 3.2: Interaktionsübersicht - Authserver & Resourceserver

Die Interaktion beginnt mit einer **API-Anfrage**, die durch die Webapplikation gestellt wird.
Der **Router** bestimmt dann, welcher **Controller** und welche Methode angesprochen werden soll. Dabei kann der Controller beliebige Services anbinden und mithilfe der Elemente der Domains unteranderem mit der Datenbank kommunizieren oder auch Schemen für die Rückgabe von Objekten definieren.
Der Controller sendet dann eine Antwort auf die API-Anfrage und schließt damit die Interaktion mit der Webapplikation ab.

3.2.5 Domain(s)

Eine Domaine ist eine Gruppierung von Interfaces, Klassen, etc. die ein Objekt beschreiben, repräsentieren oder verwalten.
Auch ein **Modell** nach dem oben beschriebenen **MVC**-Prinzips ist Teil einer solche Domaine.
Eine solche Domaine besteht meistens aus einem **Modell**, **Entity** und **Schema** Definitionen.
Dies soll in den folgenden Abschnitten am Beispiel der **Station Domaine** gezeigt werden.
Das **Entity** beschreibt dabei das interne Objekt und dessen Attribute. Ein solches Entity, wie zum Beispiel die **StationEntity** im nachfolgenden Schaubild 3.7, wird durch ein **TS**-Interface repräsentiert bzw. beschreiben.

Dabei kann angegebenen werden, welches Attribut z.B optional ist, oder welche Datentypen ein Attribut annehmen kann.

```
...
interface StationEntity {
    id: number,
    id_secure: string,
    name: string
}
...
```

Codeausschnitt 3.7: Domain - Enitybeispiel: StationEntity

Ein **Schema** dient zum Beispiel im Falle des **ResponseSchema** in Schaubild 3.8 dazu, das interne Objekt, das nach außen über die **API**-Schnittstelle bereitgesteltl wird, in seinen Attributen zu limitieren und Werte zu formatieren.

```
...
class StationSchema extends ResponseSchema{
    constructor() {
        super();

        this.fields = {
            id: [],
            id_secure: [ResponseSchemaFieldToNull],
            name: []
        }
    }
}
...
```

Codeausschnitt 3.8: Domain - Schemabeispiel: StationSchema

Ein Modell wiederrum bietet die Möglichkeit bereits vordefinierte **CRUD** & QueryBuilder-Methoden zu erweitern. So wurde zum Beispiel das **Station Modell** 3.9 um zwei Methoden erweitert, mit denen zusätzlich zu den übergebenen Information noch ein zufällige "PrivateId" generiert wird, die dann mit den anderen Daten zusammen in der Datenbank gespeichert werden.

```
...
const createPrivateId = () => {
    return
        Math.random().toString(36).substring(2, 15) +
```

```
5           Math.random().toString(36).substring(2, 15);
6   };
7
8   export default function (db?: Knex) {
9       const model = builder('stations',db);
10
11      const createStation = (data: object) => {
12          data.private_id = createPrivateId();
13          return model._create(data);
14      };
15
16      const createStations = (data: any) => {
17          ...
18      };
19
20      return {
21          ...model,
22          createStation,
23          createStations
24      }
25  };
26  ...
```

Codeausschnitt 3.9: Domain - Modellbeispiel: Stationmodell

3.2.6 Router

Mithilfe der Middelware des Express.js Frameworks wurde erreicht, dass z.B je nach Authentifizierungstoken, der assozierte **Benutzer** & **seine Berechtigungen** mithilfe des Modells einer **Domaine** geladen werden oder nicht.
Was man an der Middelware **checkAuthenticated** in Abschnitt 6.8 gut erkennen kann ist, dass abhängig davon, ob ein gültiger Authenfizierungs-Token bei der Anfrage mitgegeben wird oder nicht, der Wert der Attribute **user** & **permissions** des **req**-Objekt varieren kann.
Je nach Gültigkeit des Tokens wird auch hier der Antwort-Zyklus der Anwendung unterbrochen, um eine Fehlermeldung an den Benutzer zurückzugegeben, wie in dem Ausschnitt 6.9 zu sehen.

Das Request Objekt wird zudem um einen **user**- & **ability**-Attribut erweitert, auf die in nachfolgenden Middlewares oder allgemeinen Routen zugegriffen werden und überprüft werden kann, ob der Benutzer angemeldet ist und wenn ja über welche Berechtigungen er verfügt.

Somit kann mit einer relativ weiteren trivialen zusätzlichen Middleware, die **forceLoggedIn** 6.9 genannt wird, bei festgelegten Routen als explizite Middleware angegeben werden, um zu erzwingen, dass der Benutzer angemeldet ist.

3.2.7 Service(s)

Service(s) sind neben den **Domains** die zentralen Elemente, die von einem **Controller** angesprochen werden. Oftmals wird von einem **Controller** der Datensatz zu einem Objekt über einen **Service** manipuliert und nicht direkt über das **Modell** der **Domaine**.

Analog zu der **Webapplikation** wird wieder der **HTTP**-Client **Axios** für die API Kommunikation verwendet. Zentraler Unterschied ist, dass nun mithilfe **Typescript** die Abstraktion in Form von Vererbung einer Klasse mit dem Titel **APIService** 6.10 möglich ist, und nicht mehr eine Instanz für die jeweilige **API** Schnittstellen mithilfe der **API-Factory** 6.2 erstellt werden muss.

Hier erkennt man, dass die Konfiguration für den **Axios HTTP**-Client über den Konstrukor erfolgt.
Der Konstruktor wird dann von seiner Implementierung mit der jeweiligen Konfiguration aufgerufen.

Ein Beispiel für diese Implementierung kann man in Abschnitt 6.11 für die **VaultAPI** sehen, die dazu verwendet wird zum Beispiel den **PublicKey** der für die Erstellung eines Zuges benötigt wird, in dem Speicher **"Vault"** [sitg] zu hinterlegen. Mit diesem kann über eine **API**-Restschnittstelle kommuniziert werden.

3.2.8 Controller(s)

Ein Controller ist nach dem **MVC** Pattern für die **Steuerung** (Controller), die daraus resultierende **Präsentation** (View) und Manipulation der Daten mithilfe des **Modells** (Model) verantwortlich.
Ein Controller ist in dem Kontext der Backend Appliation, wissenschaftlich betrachtet, eine Sammlung ausgelagerter Callback Funktion einer Express.js Route.

Die Sammlung eines Controllers umfasst oftmals lediglich **CRUD**-Operationen einer bestimmten **Domaine**.

Oftmals wird für die Operation, bei der Daten erhalten werden, mehrere Methoden definiert. Zum Beispiel werden im Falle des **StationController** eine Methode **getStation** definiert, mit der eine einzige Resource zurückgegeben werden kann, als auch eine Methode **getStations**, mit der eine Sammlung an Resourcen zurückgegeben werden kann, wie es in dem Codeausschnitt 6.12 exemplarisch dargestellt wird.

3.3 Nutzerevaluation

3.3.1 Aufgabe

Die **Nutzerevaluation** fand mit zwei zentralen Komponenten statt. Zum einen die **Umfrage**, die mit Limesurveyeinem Opensource Online-Umfrage-Applikation realisiert wurde und zum anderen aus dem **Central-User-Interface**.

Initial hat jeder Teilnehmer der Evaluation eine E-Mail mit folgendem Inhalt erhalten:

```
Hallo {FIRSTNAME} {LASTNAME},

Vielen Dank, dass Sie an der Nutzerevaluation zum Central User
Interface des Personal Health Train (PHT) im Rahmen meiner
Bachelorarbeit teilnehmen.
Die Aufgaben der Nutzerevaluation können Sie der Umfrage entnehmen,
die vor Besuch des Interfaces gestartet werden soll.
Die Teilname und Durchführung ist über
folgenden Link einmalig möglich: {SURVEYURL}.

Das Interface ist unter folgender URL zu erreichen:
https://pht.tada5hi.net

Dort können Sie sich mit folgenden Anmeldedaten anmelden:
Bentutzer: {FIRSTNAME}.{LASTNAME}
Passwort: {TOKEN}

Mit freundlichen Grüßen,

{ADMINNAME} ({ADMINEMAIL})
```

Nachdem der Nutzer mithilfe der {SURVEYURL} an der Umfrage teilnehmen konnte, wurde ihm nach der Einleitung die Aufgabe gestellt, einen **Antrag**

3.3 zu erstellen. Dieser Antrag stellt die Grundlage für einen **Zug** dar.
Diesem Antrag liegen folgende Auswahlmöglichkeiten vor:

- Bei dem **Titel** handelt es sich um ein Textfeld, dass dazu dient, den Antrag bzw. das Ziel mit einem Titel zur Identifikation zu versehen.
- Das **Master Image** ist Grundlage und Standartwert für den Entrypoint, der beim Starten des Discovery- & Analysezuges benötigt wird und dem Algorithmus zugrunde liegt. Dieser Wert kann aber bei der Zugerstellung jederzeit neu gewählt werden.
- Die Skala des **Risiko** ermöglicht es anzugeben, wie hoch man selber das Risiko für die Krankenhäuser auf einer Skala einschätzt.
- Die **Risikobewertung** hat den weiteren Nutzen die Einschätzung noch in Worten schildern zu können.
- Bei dem Mehrfachauswahlfeld **Krankenhäuser** kann angegeben werden, auf welchen Stationen der Discovery- & Analysezug operieren sollen. Nach Aufgabenstellung mussten hier **Stationen 1-5** ausgewählt werden.
- Das Textfeld **angeforderte Daten/Parameter** dient dazu, zu beschreiben, auf welchen Daten der Algorithmus innerhalb der Krankenhäuser operiert.

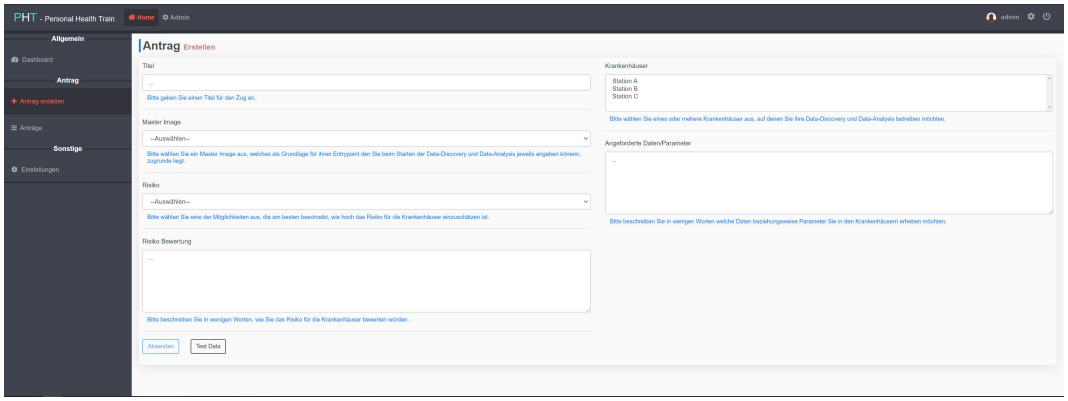


Abbildung 3.3: Screenshot des Antragfomulares im UI

Nachdem der Nutzer erfolgreich den Antrag erstellt hat, wurden ihm **drei** Fragen 3.4 dazu gestellt.

Die **1. Antrags-Frage** beschäftigte sich damit, wie viel Zeit der Nutzer für die Erstellung des Antrages benötigt hat.
Dazu wurde er gebeten, zu Beginn und Beendigung die Zeitmessung zu starten

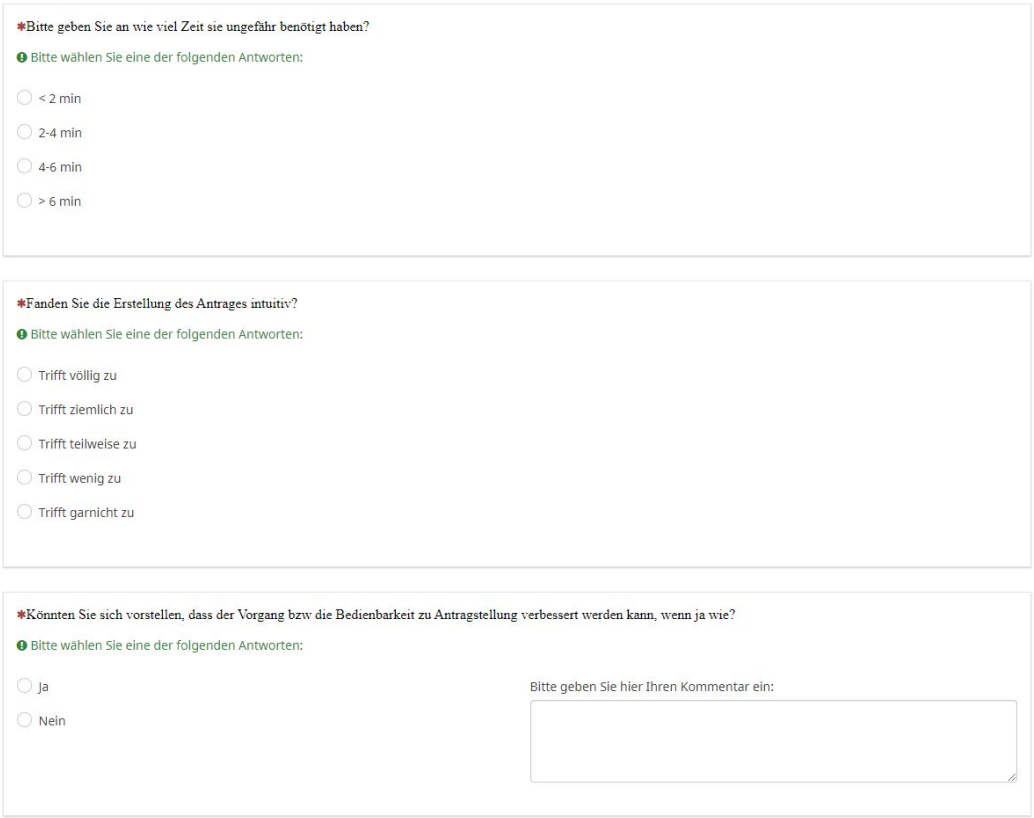


Abbildung 3.4: Screenshot der Fragen der Nuzterevaluation zu Erstellung eines Antrages

& stoppen und die Zeit im Rahmen der Auswahlmöglichkeiten anzugeben.

Das Ziel der **2. Antrags-Frage** war es herauszufinden, ob die Antragerstellung "intuitiv" war und auch ohne Vorwissen zu Thematik die Aufgabe einfach bearbeitet werden konnte.
Bei der **letzten Frage** war das Ziel herauszufinden, wie der Antrag aus verschiedenen Perspektiven verbessert werden kann.

Die letzte Aufgabe hatte zwei Teile, wobei bei beiden Teilen ein Zug erstellt werden musste.
Die Zugerstellung läuft nach genau gleichem Schema, allerdings muss zu Beginn des **TrainBuilder-Wizards**, wie in Abbildung 3.5 zu sehen, der **Zugtype** spezifiziert werden, der angibt, ob es sich um einen Discovery-Zug oder Analyse-Zug handelt.

In dem darauffolgenden Schritt **Konfiguration** muss neben einem **Masterimage** spezifiziert werden, weches Python Packet dem Algorithmus, der

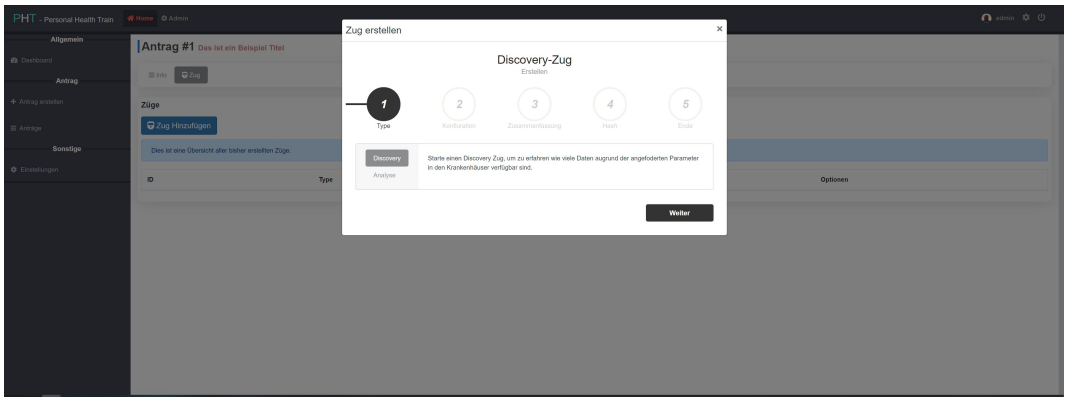


Abbildung 3.5: Screenshot des "Trainwizards" zur Erstellung eines Zuges

mit dem Multiupload-Formularfeld für **Dateien** hochgeladen werden kann, zugrunde liegt.
Daneben kann auch noch eine Subselektion der im Antrag bereits genehmigten Stationen vorgenommen werden. Dies dient speziell dazu, dass Stationen, die während der Discovery-Phase nicht relevant für eine Analyse sind, bei der Erstellung eines Analyse-Zuges weggelassen werden können.

3.3.2 Nutzergruppen

Für die Nutzerevaluation gab es zwei Gruppen. Zum einen eine **Expertengruppe**, die mit dem **PHT** vertraut sind oder zumindest in dem medizinischen Sektor angesiedelt sind und zum anderen eine **Non-Expertengruppe**, die aus Personen aus dem Bekanntenkreis entsammen.
Die Nutzergruppen bestanden jeweils aus **7** Personen und wussten dabei nicht, welche andere Personen in ihrer oder der anderen Nutzergruppe sind. Die Assoziation der Umfrageergebnisse zu der jeweiligen Nutzergruppe erfolgte über den individuellen Token, der der Person per Mail zugesendet wurde. Dieser Token war mit der Person assoziert.

Kapitel 4

Ergebnisse

4.1 Nutzerevaluation

Die zwei bereits genannten Nutzergruppen von jeweils **7** Personen konnten im Zeitraum vom **1.07.2020-26.07.2020** mithilfe des Online Umfragetools Limesurvey an der Nutzeramfrage zu dem Central User Interface teilnehmen.

4.1.1 Aufgabe 1 - Antrag erstellen

Frage 1

Bei der einführenden Bitte **"Bitte geben Sie an wie viel Zeit Sie ungefähr benötigt haben"** der **1. Aufgabe**, bei der ein Antrag erstellt werden musste, war nur eine Antwort möglich. 9 von 14 der Befragten aus beiden Nutzergruppen konnten den Antrag in unter "2 min" erstellen, 4 Personen zwischen "2-4 min" und 1 Person hat mehr als "6min" min benötigt, wie in dem Diagramm in Abbildung 4.1 zu sehen ist.

Der Ausreißer mit einer angegebenen Zeit von über "6 min", als auch 3 der 4 Befragten, die mit "2-4 min" geantwortet haben, kamen wie in Grafik 4.2b zu sehen ist, aus der NonExpertengruppe.

Frage 2

Bei der zweiten Frage **"Fanden Sie die Erstellung des Antrages intuitiv?"** war es Ziel herauszufinden, ob der Nutzer durch Darstellung und Interaktion des Central-User-Interface in der Lage ist, einen Antrag zu

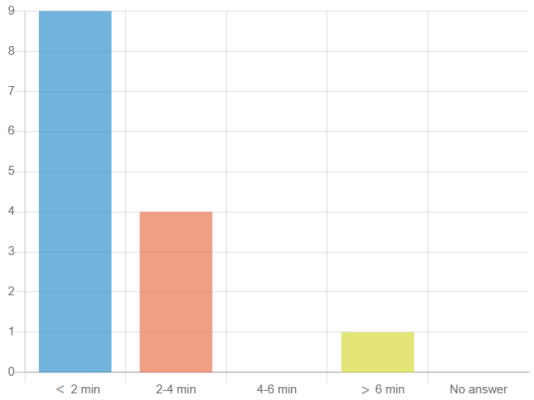
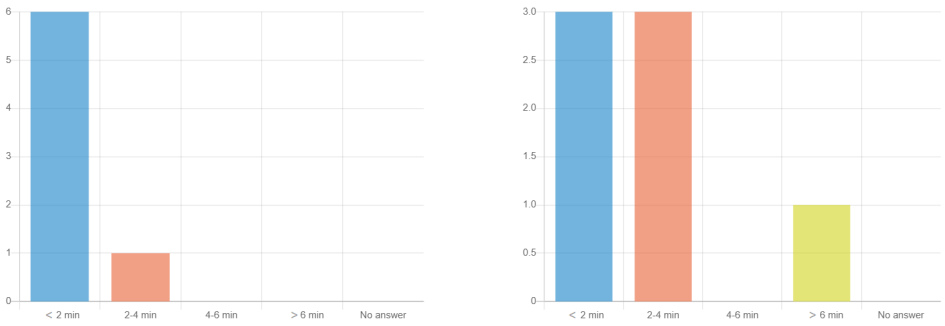


Abbildung 4.1: Aufgabe 1 - 1. Frage: "Bitte geben Sie an wie viel Zeit sie ungefähr benötigt haben?"



(a) **(b)**

Abbildung 4.2: Aufgabe 1 - 1. Frage: Experten (4.2a) & NonExperten (4.2b)

erstellen.
Es konnte wieder nur eine Antwort gegeben werden.

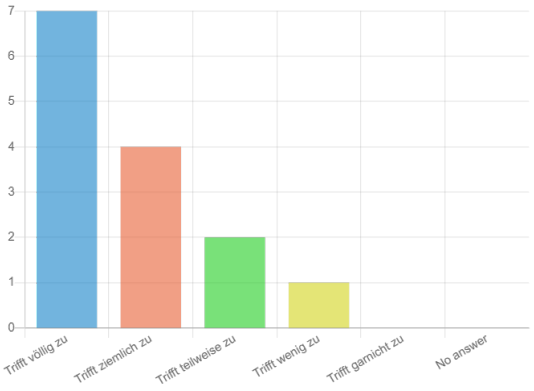
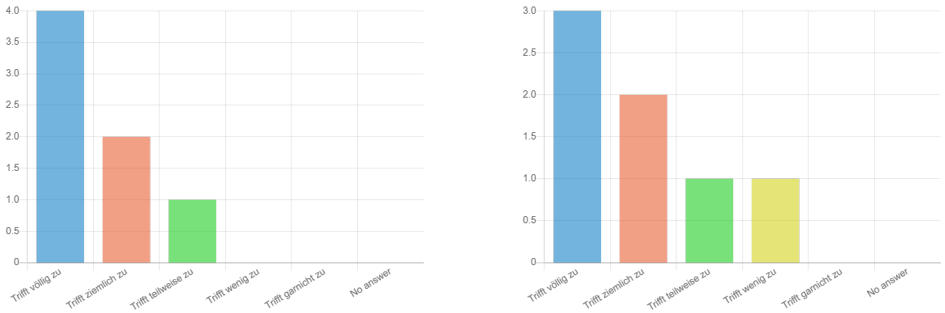


Abbildung 4.3: Aufgabe 1 - 2. Frage: "Fanden Sie die Erstellung des Antrages intuitiv?"

Hierbei antworteten 7 der insgesamt 14 Befragten "Trifft völlig zu" und 4 Personen "Trifft ziemlich zu".
Von den 4 Personen kommen dabei jeweils 2 aus der Experten- & NonExpertengruppe, wie in den beiden Grafiken in Abbildung 4.4 zu erkennen ist. Ebenso hat jeweils eine Person aus den beiden Nutzergruppe mit "Trifft teilweise zu" geantwortet. Eine Person aus der NonExpertengruppe hat "Trifft wenig zu" ausgewählt.



(a) **(b)**

Abbildung 4.4: Aufgabe 1 - 2. Frage: Experten (4.4a) & NonExperten (4.4b)

Frage 3

Auf die letzte Frage **"Könnten Sie sich vorstellen, dass der Vorgang bzw die Bedienbarkeit zu Antragstellung verbessert werden kann, wenn ja wie?"** haben 10 der 14 Personen bejaht.
Dabei kommen jeweils 5 Personen aus der gleichen Nutzergruppe. Die Verteilung der Nutzergruppenzugehörigkeit ist auch bei den anderen 4 Personen, die sich keine konkrete Verbesserung der Bedienbarkeit vorstellen können, identisch.

4.1.2 Aufgabe 2.1 - Discovery-Zug erstellen

Die erste Teilaufgabe der zweiten Aufgabe befasste sich mit der Erstellung eines Discovery-Zuges.
Dabei wurden folgende Fragen gestellt:

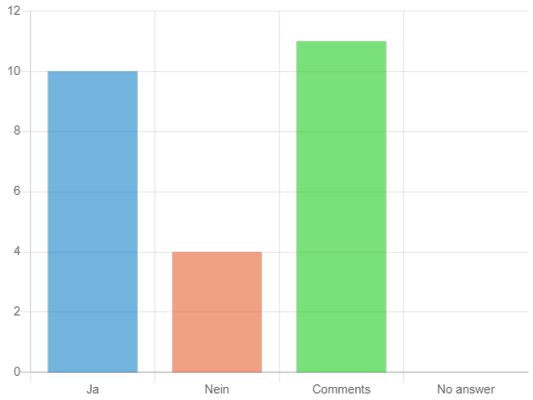
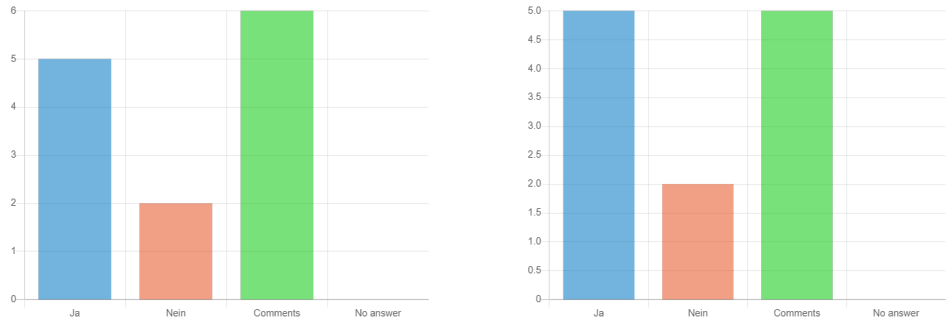


Abbildung 4.5: Aufgabe 1 - 3. Frage: "Könnten Sie sich vorstellen, dass der Vorgang bzw die Bedienbarkeit zu Antragstellung verbessert werden kann, wenn ja wie?"



(a) **(b)**

Abbildung 4.6: Aufgabe 1 - 3. Frage: Experten (4.6a) & NonExperten (4.6b)

Frage 1

Die einleitende Bitte **"Bitte geben Sie an wie viel Zeit sie ungefähr benötigt haben?"** widmete sich ebenfalls wie bereits bei der vorherigen Aufgabe 1 der benötigten Zeit die zur Durchführung der Aufgabe notwendig war.

Auch bei der zweiten Frage, benötigten die Teilnehmer für die Durchführung im Schnitt ähnlich lang wie bei der 1. Aufgabe, jedoch war das Mittelfeld diesmal deutlich stärker vertreten und niemand hat mehr als "6 min" gewählt. zu". Es haben 8 der insgesamt 14 Befragten weniger als "2min" angegeben. Eine geringere Personenzahl hat "2-4min" (4 Personen) und "4-6 min" (2 Personen) gewählt.
Die Befragten, die "2-4min" gewählt haben, kamen zu gleichen Anteilen aus der Experten- & NonExperten-gruppe, wie in Abbildung 4.8a & 4.8b zu sehen

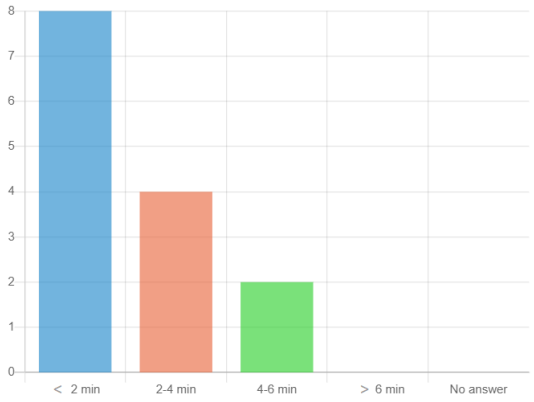
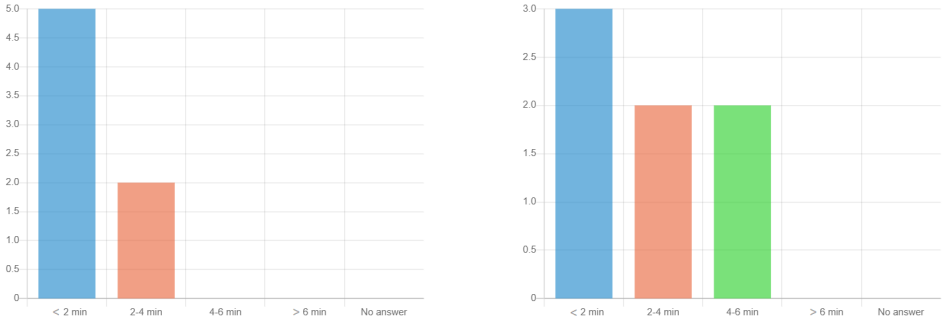


Abbildung 4.7: Aufgabe 2.1 - 1. Frage: "Bitte geben Sie an wie viel Zeit sie ungefähr benötigt haben"



(a) **(b)**

Abbildung 4.8: Aufgabe 2.1 - 1. Frage: Experten (4.8a) & NonExperten (4.8b)

ist. Die Person, die für die Durchführung mehr als "6min" benötigt hat, kam wie in Abbildung 4.8b dargestellt aus der NonExpertengruppe.

Frage 2

Auf die zweite Frage **"Fanden Sie die Erstellung des Zuges intuitiv?"** wurde am häufigsten "Trifft völlig zu" (8) und "Trifft ziemlich zu" (4) angegeben, während in gleichem Maße für "Trifft teilweise zu"(1) und "Trifft wenig zu"(1) gewählt würde, wie in der Grafik in Abbildung 4.9 zu sehen ist.

Wie in der Grafik 4.10a zu sehen, stimmten aus der Expertengruppe 3 Personen jeweils für "Trifft völlig zu" & "Trifft ziemlich zu". Nur eine Person hat "Trifft wenig zu" gewählt.
Bei der Nonexpertengruppe (Grafik 4.10b) hingegen, haben 5 Personen "Trifft ziemlich zu" gewählt und jeweils nur eine Person "Trifft ziemlich zu" & "Trifft teilweise zu".

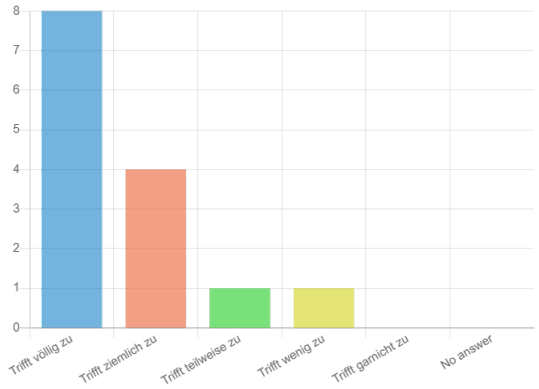
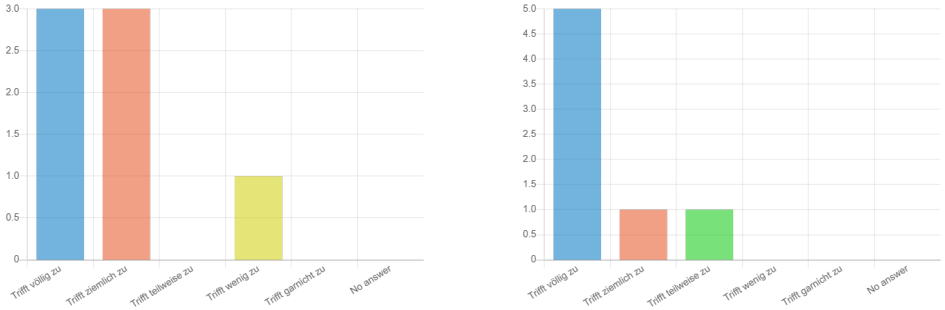


Abbildung 4.9: Aufgabe 2.1 - 2. Frage: "Fanden Sie die Erstellung des Zuges intuitiv?"



(a) **(b)**

Abbildung 4.10: Aufgabe 2.1 - 2. Frage: Experten (4.10a) & NonExperten (4.10b)

Frage 3

Bei der Frage **"Könnten Sie sich vorstellen, dass der Vorgang bzw die Bedienbarkeit zu Zugerstellung verbessert werden kann, wenn ja wie?"** konnten sich lediglich 6 von 14 Personen vorstellen die Bedienbarkeit der Zugerstellung zu verbessern (Grafik 4.11).

Dabei waren lediglich 2 Personen der Expertengruppe zugeordnet, wie in Garfik 4.12b zu sehen ist, aber der Großteil kam aus der NonExpertengruppe (4 Personen).

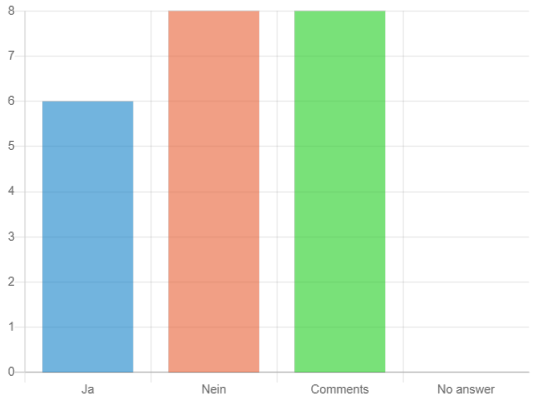
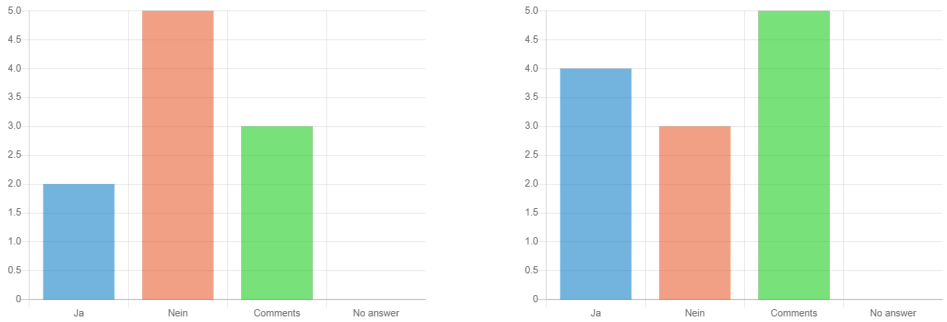


Abbildung 4.11: Aufgabe 2.1 - 3. Frage: "Könnten Sie sich vorstellen, dass der Vorgang bzw die Bedienbarkeit zu Zugerstellung verbessert werden kann, wenn ja wie?"



(a) **(b)**

Abbildung 4.12: Aufgabe 2.1 - 3. Frage: Experten (4.12a) & NonExperten (4.12b)

4.1.3 Aufgabe 2.2 - Analyse-Zug erstellen

Ähnlich wie bei Teilaufgabe 2.2 in Abschnitt 4.1.2 musste ein Zug erstellt werden. Allerdings handelte es sich diesmal um einen Analyse-Zug.

Frage 1

Bei der ersten Frage **"Bitte geben Sie an wie viel Zeit sie ungefähr benötigt haben?"** haben fast alle Teilnehmer, bis auf einen Befragten, weniger als "2 min" angegeben.
Nur ein Teilnehmer hat mehr als "6 min" benötigt.

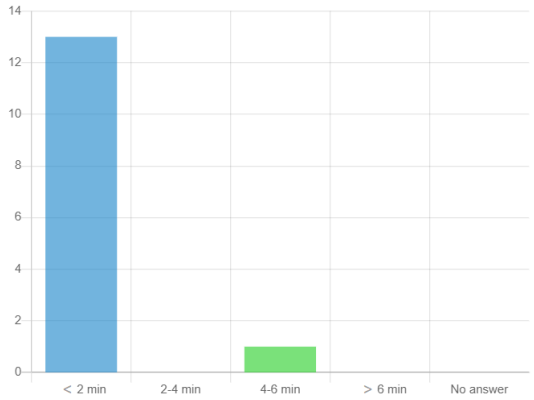


Abbildung 4.13: Aufgabe 2.2 - 1. Frage: "Bitte geben Sie an wie viel Zeit sie ungefähr benötigt haben"

Frage 2

Auf die zweite Frage **"Fanden Sie die Erstellung des Zuges intuitiv?"** hat der Großteil "Trifft völlig zu" (10) gewählt, während der Umfrage zufolge 2 Personen "Triftt ziemlich zu" und jeweils ein Befragter aus jeder Nutzergruppe "Trifft teilweise zu"(1) und "Trifft wenig zu" (1) gewählt hat, wie in der Grafik in Abbildung 4.4 zu sehen ist.

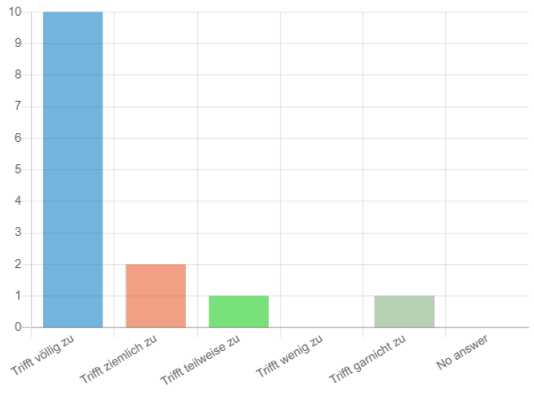
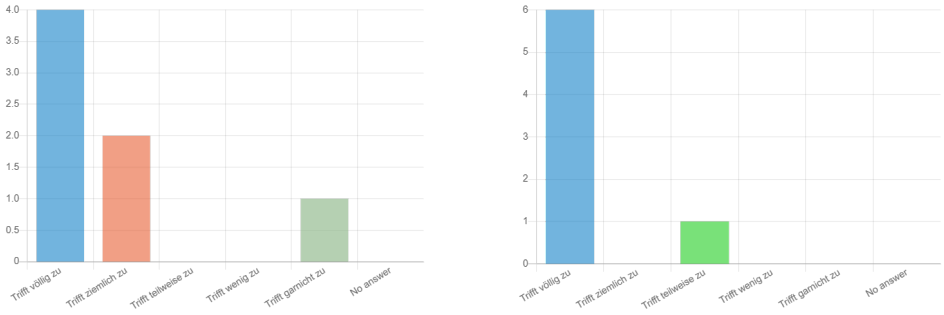


Abbildung 4.14: Aufgabe 2.2 - 2. Frage: "Fanden Sie die Erstellung des Zuges intuitiv?"

Wie in der Grafik 4.15b zu sehen ist, wählten fast alle Teilnehmer aus der Nonexpertgruppe "Trifft völlig zu", wohingegen einer "Trifft teilweise" wählte.



(a) **(b)**

Abbildung 4.15: Aufgabe 2.2 - 2. Frage: Experten (4.15a) & NonExperten (4.15b)

Frage 3

Bei der Frage **"Könnten Sie sich vorstellen, dass der Vorgang bzw die Bedienbarkeit zu Zugerstellung verbessert werden kann, wenn ja wie?"** konnten sich lediglich 6 von 14 Personen vorstellen die Bedienbarkeit der Zugerstellung zu verbessern (Grafik 4.16).

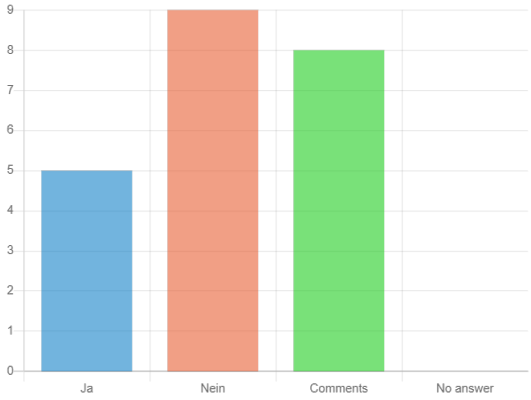
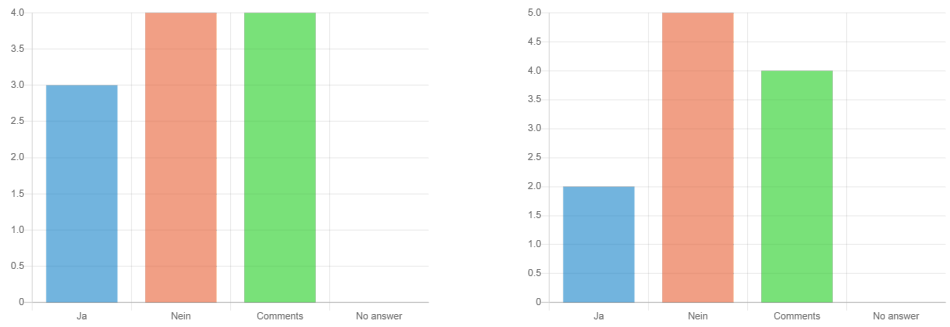


Abbildung 4.16: Aufgabe 2.2 - 3. Frage: "Könnten Sie sich vorstellen, dass der Vorgang bzw die Bedienbarkeit zu Zugerstellung verbessert werden kann, wenn ja wie?"

Dabei waren davon lediglich 2 Personen der Expertengruppe zugeordnet, wie in Garfik 4.17b zu sehen ist. Aber der Großteil der Befragten kam aus der NonExpertengruppe (4 Personen).



(a) **(b)**

Abbildung 4.17: Aufgabe 2.2 - 3. Frage: Experten (4.17a) & NonExperten (4.17b)

Kapitel 5

Diskussion

5.1 Nutzerevaluation

5.1.1 Aufgabe 1 - Antrag erstellen

Was man bei der **ersten** Frage der 1. Aufgabe bereits gut erkennen kann ist, dass fast alle Teilnehmer der Expertengruppe den Antrag in unter "2min" (6/7 Personen) erstellen konnten. Nur ein Teilnehmer hat "2-4 min" benötigt. Bei der Nonexpertengruppe konnte nur die Hälfte im Vergleich zu den Experten den Antrag in unter "2min" erstellen. Zwei Personen mehr als bei der Expertengruppe waren auch für den Zeitraum "2-4 min" zu vermerken.
Dies liegt vermutlich unteranderem an den Informationen, die bei der Antragerstellung angegeben werden mussten. Zudem könnte der Kontext des Antrags nicht ganz eindeutig gewesen sein, aber das war anzunehmen.

Auch bei der **zweiten** Frage fanden die Experten mit knapper Mehrheit die Antragerstellung intuitiver als die NonExperten.
Bei der letzten Frage der 1. Aufgabe konnten sich Befragte, zu gleichen Teilen aus der Experten- und NonExpertengruppe vorstellen, wie man die Erstellung eines Antrages verbessern könnte.

5.1.2 Aufgabe 2.1 - Discovery-Zug erstellen

Bei der Zugerstellung konnte die Expertengruppe wieder im Schnitt die Aufgabe deutlich schneller lösen. Fünf der sieben Befragten haben weniger als "2 min" benötigt und zwei Personen "2-4 min". In der NonExpertengruppe waren dies nur drei & zwei Personen. Zudem konnten zwei Personen in der Nonexpertengruppe die Aufgabe erst in "4-6 min" lösen.

Die Zugerstellung fanden beide Gruppen relativ gleich intuitiv. In bei-

den Gruppen gaben 6 der 7 Befragten dabei "Trifft völlig zu" oder "Trifft ziemlich zu".

Bei der letzten Frage, wie der Vorgang bzw die Bedienbarkeit zu Antragstellung verbessert werden kann, konnten doppelt so viele Personen aus der Nonexpertengruppe als in der Expertengruppe vorstellen.
Dies könnte an dem Blick eines Ausstehenden liegen, da dieser mit der Thematik noch nicht so vertraut ist und daher mehr Möglichkeiten sehen könnte, wie der Vorgang verbessert werden kann und schlussendlich auch für Ausstehende verständlicher wird.

Dies spiegelt sich auch in den Kommentaren wie z.B *"Ich kann mir vorstellen, dass viele Fachfremde von den vielen Fachbegriffen abgeschreckt sind. Zb wizard, hash, publicKey können schwer vorausgesetzt werden, allerdings war es mit der Anleitung und den download/uploadbaren files gut machbar. Falls möglich, Nutzung von Fachbegriffen in der Anleitung noch weiter reduzieren, denke ich. Das hochladen eines python scripts ist irgendwie unintuitiv, evtl kurzer Satz dazu wozu?"* wieder.

5.1.3 Aufgabe 2.2 - Analyse-Zug erstellen

Die letzte Teilaufgabe, die sich wie zuvor um eine Zugerstellung handelte, allerdings in diesem Fall um einen Analyse-Zug.
Wie erhofft hat sich die Quote hinsichtlich der Geschwindigkeit zum Lösen der Aufgabe im Vergleich zu davor verbessert.
In der Expertengruppe konnten nun alle sieben Befragten die Aufgabe in unter "2 min" lösen. Auch bei den Experten waren dies nun 6 von 7 Personen. Nur noch eine Person von zwei hat "4-6 min" zum Bearbeiten der Aufgabe benötigt.

Dies könnte daran liegen, dass in der Expertengruppe noch viele unterschiedliche Vorstellungen existieren, wie ein Zug konkret spezifiziert & aufgebaut sein sollte.

Die Zugerstellung fanden wieder beide Gruppen relativ gleich intuitiv.

Wie die Erstellung eines Analyse-Zuges verbessert werden könnte, können sich aus der NonExpertengruppe nun weniger Teilnehmer vorstellen als zuvor. Dies könnte daran liegen, dass der Unterschied zwischen den beiden Zügen nicht ganz klar ist oder warum die Notwendigkeit dafür besteht. Bei den Experten hingegen ist es eine Stimme mehr geworden, die sich vorstellen könnte, wie der Vorgang besser werden könnte.

Der Verdacht warum die Notwendigkeit dafür besteht, dass Disovery- &

Analyse-Züge differenziert werden müssen, taucht auch in einem Kommentar eines Experten auf: *"Warum muss hier im Wesentlichen nochmal die gleiche Information eingegeben werden wie bei dem Discovery train? Ich halte Discovery-Trains nach wie vor für eher unnötig. Entweder es sind separate Züge, dann ist keine Unterscheidung notwendig - so ist es jetzt. Oder es ist *ein* Zug, der im Discovery- oder Production-Mode läuft, dann kann man sich die Doppeleingabe sparen. So macht die Modusausawahl für mich keinen Sinn."*.

Grund für die Differenzierung war es, die beiden Phasen (Discovery & Analyse) voneinander zu trennen. Da bei der Discovery Phase nur Meta Daten gesammelt werden und keine Modelle trainiert werden, welche deutlich mehr Zeit zur Durchführung bei einer Station beanspruchen würden, kann ein Discovery-Zug zum Beispiel priorisiert in der Warteschlange zur Ausführung einer Station behandelt werden.
Die Idee des Experten für einen Zug im "Disocvery- oder Production-Mode" hört sich nach einer guten Lösung dafür an, da so nur ein Zug erstellt werden müsste und dieser auf verschieden Weisen gestartet werden könnte.
Grundsätzlich können aber aber nach Genehmigung eines Antrages beliebig viele Züge erstellt werden.

5.1.4 Fazit

Ein Ziel der Nutzerevaluation war es herauszufinden, ob es möglich ist, Personen, die mit der Thematik nicht sehr oder gar nicht vertraut sind, dazu in die Lage versetzen zu können, einen Antrag und auch ein daraus möglichen Zug zu erstellen.
Trotz wenig Vorwissen konnten die Nonexperten wie auch die Experten die Aufgaben in schnellem Tempo durchführen.
Die Erstellung eines Antrages als auch eines Zuges fanden beide Gruppen zu großen Teilen intuitiv. Allerdings konnten sich auch aus beiden Gruppen viele Personen vorstellen wie der Antrag- & Zugerstellung verbessert werden könnte.
Dies könnte dazu beitragen, mögliche Fehler die aus der Bedienbarkeit, Darstellung und Bereitstellung an Informationen resultieren könnten, zu vermeiden und das Central-User-Interface im weiteren Verlauf zu verbessern.

Kapitel 6

Appendix

6.0.1 Methoden

Frontend - Webapplikation

```
{
    "name": "app",
    "version": "1.0.0",
    "description": "Peter Placzek Bachelor Thesis",
    "author": "Peter Placzek",
    "private": true,
    "scripts": {
        "dev": "nuxt",
        "build": "nuxt build",
        "start": "nuxt start",
        "generate": "nuxt generate",
        "lint": "eslint --ext .js,.vue --ignore-path .gitignore ."
    },
    "dependencies": {
        "@casl/ability": "^4.1.4",
        "@casl/vue": "^1.1.0",
        "@fortawesome/fontawesome-free": "^5.13.0",
        "@nuxtjs/dotenv": "^1.4.0",
        "@types/moment": "^2.13.0",
        "axios": "^0.19.2",
        "bootstrap": "^4.1.3",
        "bootstrap-vue": "^2.11.0",
        "core-js": "^2.6.5",
        "eslint-loader": "^3.0.4",
        "moment": "^2.26.0",
        "nuxt": "^2.12.2",
```

```
        "vue": "^2.6.11",
        "vue-form-wizard": "^0.8.4",
        "vue-gravatar": "^1.3.1",
        "vue-socket.io": "^3.0.7",
        "vue-upload-component": "^2.8.20",
        "vue2-dropzone": "^3.6.0",
        "vuelidate": "^0.7.5",
        "vuescroll": "^4.16.0",
        "vuex": "latest"
    },
    "devDependencies": {
        "@nuxtjs/eslint-config": "^2.0.0",
        "@nuxtjs/eslint-module": "^1.0.0",
        "babel-eslint": "^10.1.0",
        "eslint": "^6.1.0",
        "eslint-plugin-nuxt": "^0.5.2"
    }
}
```

Codeausschnitt 6.1: Konfigurationsdatei - package.json

```
...
const apiFactory = {
    create() {
        let instance = axios.create();

        instance.
            defaults.
                headers.
                    common['Accept-Language'] = 'de';

        return this.init(instance);
    },
    init: (instance) => ({
        init(baseURL) {
            instance.defaults.baseURL = baseURL
        },

        // Http methods
        get(resource) {
            return instance.get(resource)
        },
        ...
    }),
```

```
    ...
};
...
```

Codeausschnitt 6.2: APIFactory

```
...
const ProposalEdgeMapping = {
    id: 'id',
    title: 'title',
    requestedData: 'requested_data',
    risk: 'risk',
    riskComment: 'risk_comment',
    masterImageId: 'master_image_id',
    userId: 'user_id',
    createdAt: 'created_at',
    updatedAt: 'updated_at',
    stationsTotal: 'stations_total',
    stationIds: 'station_ids'
};

const ProposalEdge = {
    addProposal: async (data) => {
        try {
            data = formatToEdgeRequestObject(
                data,
                ProposalEdgeMapping
            );

            const response = await ResourceApiService.post(
                'proposals',
                data
            );

            return response.data;
        } catch (e) {
            throw new Error('Der Antrag konnte' +
                            'nicht erstellt werden.');
        }
    },

    //-------------------------------------------------------------------

    getProposal: async (id) => {
```

```
        try {
            const response = await ResourceApiService.get(
                'proposals/' + id
            );

            return parseEdgeResponseObject(
                response.data,ProposalEdgeMapping
            );
        } catch (e) {
            console.log(e);
            throw new Error(
                'Der Antrag konnte nicht gefunden werden.'
            );
        }
    },
    ...
}
...
```

Codeausschnitt 6.3: ProposalEdge & ProposalMapping

```
...
const AuthService = {
    /**
     * Set authorization token header on api requests.
     * @param token
     */
    setRequestToken: (token) => {
        let key = 'Authorization';
        let value = 'Bearer ' + token;

        console.log('Set authorization header...')

        ResourceApiService.setHeader(key, value);
        AuthApiService.setHeader(key, value)
    },
    ...
}
...
```

Codeausschnitt 6.4: AuthService

```
...
const tokenKey = 'token';
```

```
...
const AuthStorage = {
    // Access Token operations

    getToken () {
        return StorageService.getJson(tokenKey);
    },
    setToken (data) {
        StorageService.setJson(tokenKey, data);
    },
    dropToken () {
        StorageService.dropJson(tokenKey);
    },
    ...
}
...
```

Codeausschnitt 6.5: AuthStorage

```
...
const actions = {
    ...
    /**
     * Try to login the user with given credentials.
     *
     * @param commit
     * @param dispatch
     *
     * @param data
     * @param provider
     *
     * @return {Promise<boolean>}
     */
    async triggerLogin (
        { commit, dispatch },
        { data, provider}
    ) {
        commit('loginRequest');

        try {
            const token = await AuthService.login(
                data,
                provider
            );
```

```
26
27              ...
28              commit('setToken', token);
29
30              AuthApiService.
31                  mountInterceptor('auth');
32
33              ResourceApiService.
34                  mountInterceptor('auth');
35
36              dispatch('triggerRefreshMe');
37          } catch (e) {
38              if (e instanceof AuthenticationError) {
39                  commit('loginError', {
40                      errorCode: e.errorCode,
41                      ...
42                  })
43              }
44
45              throw new Error(e.message);
46          }
47      },
48      /**
49       * Try to trigger user refresh.
50       *
51       * @param commit
52       * @param dispatch
53       *
54       * @returns {Promise<boolean>}
55       */
56      async triggerRefreshMe ({ commit, dispatch }) {
57          ...
58      },
59      ...
60  };
61  ...
```

Codeausschnitt 6.6: AuthStore - Aktionen

```
1  ...
2  export default ({ app, store, env }, inject) => {
3      ...
4      const permission = {
5          has: function (name) {
```

```
            try {
                getPermissionByName(
                    store.state.auth.permissions,
                    name
                );
                return true;
            } catch (e) {
                return false;
            }
        },
        can: function() {
            return ability.can(...arguments);
        },
        getPower: function() {
            return getPermissionPowerByName(
                store.state.auth.permissions,
                'power',
                ...arguments
            );
        },
        getPowerInverse: function() {
            return getPermissionPowerByName(
                store.state.auth.permissions,
                'powerInverse',
                ...arguments
            );
        }
    }

    inject('permission',permission);
    app.$permission = permission;
}
...
```

Codeausschnitt 6.7: AuthPlugin

Backend - Authservcer & Resourceserver

```
...
const checkAuthenticated = async (
    req: any,
    res: any,
    next: any
```

```
) => {
    let { authorization } = req.headers;

    let userObject: UserEntity | null = null;
    let userId: number | null | undefined = null;

    if(typeof authorization !== "undefined") {
        let provider;

        let token = authorization.split(" ")[1];

        switch (AuthConfig.lapMode) {
            case AuthLAPMode.LAPBasic:
                provider = new TokenProvider();

                if(
                (token = await provider.verifyToken(token))
                ===
                false
                ) {
                    res.cookie('token', null, {
                        maxAge: Date.now()
                    });

                    return res._failUnauthorized({
                        message: 'Der Token ist nicht gueltig.',
                        errorCode: 'invalid_token'
                    });
                }

                userId = token.id;
                break;
            case AuthLAPMode.LAPOauth2:
                return res._failServerError({
                    message: 'Oauth2 nicht implementiert....',
                    errorCode: 'not_implemented'
                });
        }

        if(typeof userId === 'undefined' || !userId) {
            return res._failUnauthorized({
                message: 'Es konnte kein Benutzer' +
                         ' assoziert werden.'
```

```
            });
        }

        let query = UserModel()._find({
            id: userId
        });

        try {
            let {password, ...user} =
                <AuthUserEntity> await onlyOneRow(query);
            userObject = <UserEntity> user;

        } catch (e) {
            return res._failUnauthorized({
                message: 'Der Benutzer existiert nicht mehr.'
            });
        }
    }

    req.user = userObject;
    req.ability = await AuthorizationService
                            .defineAbilityFor(userId);
    next();
};
...
```

Codeausschnitt 6.8: AuthMiddleware - Check

```
...
const forceLoggedIn = async (req: any, res: any, next: any) => {
    if(req.user == null) {
        res._failUnauthorized({
            message: 'Sie muessen angemeldet sein.'
        });
        return;
    }

    next();
};
...
```

Codeausschnitt 6.9: AuthMiddleware - Force

```
...
```

```
class APIService {
    private api: AxiosInstance;

    /**
     * API Service
     *
     * @param config
     */
    constructor(config: ApiRequestConfig) {
        this.api = axios.create(config);
        ...
    }

    ...
}
...
```

Codeausschnitt 6.10: AuthMiddleware - Force

```
...
class VaultAPI extends APIService {
    /**
     * Vault Token
     */
    private readonly token: string;

    //-----------------------------------------------

    /**
     * Constructor
     */
    constructor () {
        let config: ApiRequestConfig = {
            withCredentials: true,
            timeout: 3000,
            baseURL: '',
            token: ''
        };
    }
    ...
}
...
```

Codeausschnitt 6.11: VaultAPIService

```
...
const getStations = async (req: any, res:any) => {
  let stations = await StationModel()._findAll();

  let stationSchema = new StationSchema();
  stations = stationSchema.applySchemaOnEntities(stations);

  res._respond({data: stations});
};

const getStation = async (req: any, res: any) => {
  let {id} = req.params;

  if(!id) {
    return res._failNotFound();
  }

  let station = await StationModel()._findOne(id);

  if(!station) {
      return res._failNotFound();
    }

    let stationSchema = new StationSchema();
    station = stationSchema.applySchemaOnEntity(station);

  res._respond({data: station});
};

...
```

Codeausschnitt 6.12: StationController - getStation & getStations Beispiel

Literaturverzeichnis

[Bem] Joshua Bemenderfer. Understanding vue.js lifecycle hooks. https://www.digitalocean.com/community/tutorials/vuejs-component-lifecycle.

[Fou] Node.js Foundation. Node.js. https://nodejs.org/.

[Hau] Philipp Hauer. Das decorator design pattern. https://www.philipphauer.de/study/se/design-pattern/decorator.php.

[Lud] Simon Ludwig. One-way-data-binding, two-way-data-binding und virtual dom diff. https://frontend.namics.com/2016/01/18/one-way-data-binding-two-way-data-binding-und-virtual-dom-diff/.

[Pet00] Lars Peterke. *Vue.js kurz gut*, page 51. OReilly, 2100.

[sita] axios. https://github.com/axios/axios.

[sitb] Container image registry - harbor. https://goharbor.io/.

[sitc] Javascript. https://de.wikipedia.org/wiki/JavaScript.

[sitd] Middleware für die verwendung in express-anwendungen schreiben. https://expressjs.com/de/guide/writing-middleware.html.

[site] Node.js. https://de.wikipedia.org/wiki/Node.js.

[sitf] Pht manifest. https://www.dtls.nl/wp-content/uploads/2017/12/PHT$_M anifesto.pdf$.

[sitg] Vault. https://github.com/hashicorp/vault.

[sith] Web storage konzepte und verwendung. https://developer.mozilla.org/de/docs/Web/API/Web$_S torage_A PI$.

[Youa] Evan You. State management. https://vuejs.org/v2/guide/state-management.html.

[Youb] Evan You. Vue.js. https://vuejs.org/.

[Zim] Lukas Zimmermann. Pht Übersicht grafik. https://docs.google.com/presentation/d/13XtHjYxMWNgSMj-$_hIXbnHdoub1unOzq6toJLwN1x6U/editslide = id.g8d75cb0e62_{04}9$.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift