**HOCHSCHULE für Angewandte Wissenschaften LANDSHUT**

**UNIVERSITY of Applied Sciences**

**FAKULTÄT Informatik**

**Entwicklerdokumentation zur Anpassung des ADAMO Modelers**

Im Rahmen des Studienprojekts im Master Informatik

vorgelegt von

Masood Ahmed

Jennifer Espich

Christina Frank

Granit Gecaj

Daniel Lackmann

Markus Schmidtner

eingereicht am: 31.08.2018

Betreuer:

Daniel Hilpoltsteiner

Prof. Dr. rer. oec. Christian Seel

# Inhaltsverzeichnis

[Inhaltsverzeichnis II](#_Toc523475402)

[Abbildungsverzeichnis IV](#_Toc523475403)

[Tabellenverzeichnis VI](#_Toc523475404)

[Abkürzungsverzeichnis VII](#_Toc523475405)

[1 Allgemeine Beschreibung 1](#_Toc523475406)

[1.1 Rahmenbedingungen 1](#_Toc523475407)

[1.2 Entwicklerwerkzeuge 1](#_Toc523475408)

[2 Technischer Aufbau 3](#_Toc523475409)

[2.1 Datenbank 3](#_Toc523475410)

[2.2 MQTT 5](#_Toc523475411)

[2.3 Express.js 7](#_Toc523475412)

[2.4 Angular Frontend 7](#_Toc523475413)

[3 Installationsanleitung 8](#_Toc523475414)

[3.1 Git bzw. GitKraken 8](#_Toc523475415)

[3.2 Visual Studio Code 8](#_Toc523475416)

[3.3 Angular und express.js 9](#_Toc523475417)

[3.4 Open VPN 10](#_Toc523475418)

[3.5 PostgreSQL und pgAdmin (optional) 11](#_Toc523475419)

[4 Lokale und Server-Datenbank 12](#_Toc523475420)

[4.1 Anlage einer lokalen Datenbank 12](#_Toc523475421)

[-- Database: ipim 12](#_Toc523475422)

[-- Table: public.session 13](#_Toc523475423)

[-- Table: public.users 13](#_Toc523475424)

[-- Triggerfunction: set\_lastlogin 14](#_Toc523475425)

[-- Trigger: set\_lastlogin 14](#_Toc523475426)

[-- Table: public.model 14](#_Toc523475427)

[-- Triggerfunction: set\_lastchange() 15](#_Toc523475428)

[-- Trigger: set\_lastchange 15](#_Toc523475429)

[-- Table: public.partialmodel 15](#_Toc523475430)

[-- Table: public.permission 16](#_Toc523475431)

[-- Table: public.role 16](#_Toc523475432)

[-- Table: public.userprofile 17](#_Toc523475433)

[4.2 Wechsel zwischen lokaler und Server-Datenbank 17](#_Toc523475434)

[5 Starten des ADAMO für Entwickler 18](#_Toc523475435)

[6 Aufbau der Anwendung im VS Code 20](#_Toc523475436)

[6.1 Aufbau der GUI 20](#_Toc523475437)

[6.1.1 Login 20](#_Toc523475438)

[6.1.2 Modeller 21](#_Toc523475439)

[6.1.2.1 modellerPage 21](#_Toc523475440)

[6.1.2.2 Model 23](#_Toc523475441)

[6.1.2.3 Versionierung 24](#_Toc523475442)

[6.1.2.4 Navigation Bar 27](#_Toc523475443)

[6.1.2.5 Modeller-Palette 27](#_Toc523475444)

[6.1.2.6 Modale 29](#_Toc523475445)

[6.1.2.7 Modeller Funktionen 32](#_Toc523475446)

[6.1.3 Administration Page 40](#_Toc523475447)

[6.1.3.1 Komponenten der Administration Page 41](#_Toc523475448)

[6.1.4 Ändern des Farbschemas der Login Page 42](#_Toc523475449)

[6.1.4.1 Login 42](#_Toc523475450)

[6.1.4.2 Schnittstelle GUI – Server 43](#_Toc523475451)

[6.1.5 Dependencies für die GUI 44](#_Toc523475452)

[6.2 Aufbau des Express.js-Servers 45](#_Toc523475453)

[6.3 User.js, permission.js, model.js, partmodel.js, profile.js und role.js 45](#_Toc523475454)

[6.4 Aufbau des MQTT-Servers 46](#_Toc523475455)

[7 Tests 48](#_Toc523475456)

# Abbildungsverzeichnis

[Abbildung 1: Datenbanktabellen und ihre Beziehungen zueinander 5](#_Toc523475350)

[Abbildung 2: Publish/Subscriber Modell bei MQTT 6](#_Toc523475351)

[Abbildung 3: Starten des Mosca-Servers über die Konsole 6](#_Toc523475352)

[Abbildung 4: Starten des express.js-Servers über die Konsole 7](#_Toc523475353)

[Abbildung 5: Klonen eines Repositories mittels GitKraken 8](#_Toc523475354)

[Abbildung 6: Installation von zusätzlichen Komponenten 9](#_Toc523475355)

[Abbildung 7: npm install im API Ordner auf dem express.js-Server 9](#_Toc523475356)

[Abbildung 8: npm install im angular2-Ordner im Frontend 10](#_Toc523475357)

[Abbildung 9: Download-Möglichkeiten von Open VPN der HAW Landshut 10](#_Toc523475358)

[Abbildung 10: express.js starten 18](#_Toc523475359)

[Abbildung 11: Öffnen einer neuen Konsole 18](#_Toc523475360)

[Abbildung 12: Starten des MQTT-Servers 19](#_Toc523475361)

[Abbildung 13: Starten des Node-Paketmanagers 19](#_Toc523475362)

[Abbildung 14: Login Page im KIP-Design vs. Login Page im IPIM-Design 20](#_Toc523475363)

[Abbildung 15: Front Page Struktur in VS Code 21](#_Toc523475364)

[Abbildung 16: Einstiegsseite zum Modeller: modellerPage 22](#_Toc523475365)

[Abbildung 17: modellerPage Struktur in VS Code 22](#_Toc523475366)

[Abbildung 18: Definition von Model und User 24](#_Toc523475367)

[Abbildung 19: Versionierung von Modellen anhand eines scrum Beispiels 25](#_Toc523475368)

[Abbildung 20: Logik zur Versionierung von Modellen 26](#_Toc523475369)

[Abbildung 21: File zur Versionierung in der Dateistruktur 26](#_Toc523475370)

[Abbildung 22: Anzeige eines Modells in der Navigation Bar 27](#_Toc523475371)

[Abbildung 23: Palette mit Icons für weitere Funktionen 28](#_Toc523475372)

[Abbildung 24: Dateien zur Palettenverwaltung 29](#_Toc523475373)

[Abbildung 25: Term Modal 29](#_Toc523475374)

[Abbildung 26: Variablen Modal 30](#_Toc523475375)

[Abbildung 27: Dateistruktur zu den Modalen 31](#_Toc523475376)

[Abbildung 28: Aufruf der Komponente aus dem Variablen Modal heraus 32](#_Toc523475377)

[Abbildung 29: Komponente zum Variablen Modal enthält den zugehörigen selector 32](#_Toc523475378)

[Abbildung 30: Download eines Modells in bpmn-Format 33](#_Toc523475379)

[Abbildung 31: Download eines Modells im svg-Format 33](#_Toc523475380)

[Abbildung 32: Setzen von Variablen 34](#_Toc523475381)

[Abbildung 33: Setzen von Termen 34](#_Toc523475382)

[Abbildung 34: Evaluieren von einzelnen Modellen 35](#_Toc523475383)

[Abbildung 35: zip-Download der betroffenen Modelle beim kaskadierenden Evaluieren 35](#_Toc523475384)

[Abbildung 36: Dateistruktur zum kaskadierenden Auswerten 36](#_Toc523475385)

[Abbildung 37: Setzen eines Subprozesses 37](#_Toc523475386)

[Abbildung 38: Öffnen eines Subprozesses in einem neuen Tab 37](#_Toc523475387)

[Abbildung 39: Farbliches Hervorheben von Prozesspfaden anhand ihrer zugewiesenen Terme 38](#_Toc523475388)

[Abbildung 40: Reset des Diagramms 39](#_Toc523475389)

[Abbildung 41: Anzeige aller Prozessreferenzen 40](#_Toc523475390)

[Abbildung 42: Coding zur Administration Page 41](#_Toc523475391)

[Abbildung 43: Struktur der Komponenten für die Administration Page 41](#_Toc523475392)

[Abbildung 44: Änderung des Farbschemas der Login-Seite-Umrandung 42](#_Toc523475393)

[Abbildung 45:Änderung des Farbschemas der Login-Seite-Button 42](#_Toc523475394)

[Abbildung 46: api.service.ts verknüpft die Oberfläche mit den Datenbank-Requests 43](#_Toc523475395)

[Abbildung 47: Dokumentation der Aufrufe in api.service.ts 44](#_Toc523475396)

[Abbildung 48: Dateistruktur für die Dependencies 44](#_Toc523475397)

[Abbildung 49: getAll-Request für Model in model.js 45](#_Toc523475398)

[Abbildung 50: Aufbau des Express.js-Servers 46](#_Toc523475399)

[Abbildung 51: Aufsetzen von einem Fake-Backend 49](#_Toc523475400)

[Abbildung 52: Verwendung des Fake-Backends 49](#_Toc523475401)

# Tabellenverzeichnis

[Tabelle 1: Verbindungsdaten der lokalen Datenbank 17](#_Toc523473654)

# Abkürzungsverzeichnis

ADAMO Adaptiver Modeller

BPMN Business Process Model and Notation

CMMN Case Management Model and Notation

DMN Decision Model and Notation

IPIM Institut für Projektmanagement und Informationsmodellierung und dem

KIP Kompetenznetzwerk intelligente Produktionslogistik

MQTT Message Queue Telemetry Transport

SQL Structured Query Language

VPN Virtual Private Network

VS Visual Studio

# Allgemeine Beschreibung

Camunda stellt einen Modeller bereit, der sowohl für die Prozess- als auch die Entscheidungsmodellierung geeignet ist und die Modellierungssprache BPMN 2.0 unterstützt. Aufgrund der eingeschränkten Funktionalität und neuer Bedürfnisse von Anwendern wird dieses Werkzeug in Zusammenarbeit des Instituts für Projektmanagement und Informationsmodellierung (IPIM) und dem Kompetenznetzwerk Intelligente Produktionslogistik (KIP) zum adaptiven Modeller (ADAMO) erweitert.

Als neue Funktion zusätzlich derer von Camunda befähigt der Modeller auf BPMN basierende Prozesse ortsunabhängig, simultan und kollaborativ zu visualisieren. Hierfür wird eine Publisher/Subscriber Technologie mit Hilfe des Protokolls MQTT verwendet.

Außerdem können diese Modelle anhand eingegebener Variablen ausgewertet werden, sodass nur die für einen definierten Fall zutreffenden Teile eines Prozessmodells angezeigt werden. Dabei werden auch Subprozesse berücksichtigt.

## Rahmenbedingungen

Ziel der Lehrveranstaltung Praxisorientiertes Studienprojekt ist, komplexe Projekte zu organisieren und durchzuführen. Die Studierenden sind dafür in der Lage, wissenschaftliche, technische und soziale Kompetenzen einzusetzen. Sie haben Teamarbeit, Management und Kontrolle von Projekten, selbstständige wissenschaftliche und technische Arbeit im Team trainiert. Sie können fachübergreifende Kenntnisse anwenden und Projektergebnisse professionell präsentieren.

## Entwicklerwerkzeuge

Die Realisierung des Systems erfolgte mit Hilfe einiger Werkzeuge.

Für das Datenbankmodell wurde yEd Graph Editor verwendet.

Die Erstellung von Source-Code wurde mit der Entwicklungsumgebung Visual Studio Code (VS Code) realisiert.

Die Präsentationsschicht wurde mit Angular 2 und realisiert.

Die Applikationslogik wurde auf Basis von express.js und MQTT realisiert.

Die Datenbank wurde mit PostgreSQL bzw. dem pgAdmin4 realisiert.

Zur Koordination der Teilnehmer untereinander und auch der verschiedenen Software- Entwicklungsstufen wurde Git unter Nutzung des webbasierten Git-Hosting-Dienstes GitHub verwendet.

Die Kommunikation der Teilnehmer untereinander erfolgte neben persönlichen Treffen über die Kommunikationswerkzeuge Telegram und Skype.

# Technischer Aufbau

Der bisherige Camunda Modeler basiert auf der bpmn.io Engine. Das Projekt ist jedoch rein auf die Modellierung im Web spezialisiert und enthält weder eine kollaborative noch eine auswertende Funktionalität. Da es jedoch als freies Open-Source Projekt zur Verfügung steht, eignet es sich perfekt als Basis für die technische Implementierung der gewünschten Funktionalitäten.

Der ADAMO wird im Rahmen einer dreischichtigen Architektur, der sog. Three Tier Architecture realisiert.

Wie der Name bereits sagt, besteht diese Architektur aus drei Schichten. Angefangen mit der Datenbankschicht als unterste Ebene, sorgt diese mit dem Einsatz von PostgreSQL (kurz Postgres) für die Persistenz der Daten. Die mittlere Schicht ist die Applikationsschicht, welche zwei Server umfasst. Zum einen kommt express.js zum Einsatz. Dieser enthält Algorithmen, Regeln und Strukturen, um die Elemente des ADAMO (Modell, User, etc.) und Funktionen (anlegen, bearbeiten, löschen, etc.) der Anwendung beschreiben zu können. Zum anderen wird das Protokoll MQTT in Verbindung mit einem Mosca Server eingesetzt, welcher das Subscriben auf ein/mehrere Modelle sowie das kollaborative Arbeiten an Modellen ermöglicht. Als oberste Schicht folgt die Präsentationsschicht und beinhaltet zwei getrennte Ansichten mit graphischer Benutzungsschnittstelle.

Von diesen dient die erste Ansicht der Pflege des gesamten Datenbestands auf der Administrationsseite. Dabei erhält ein Anwender jedoch nur mit den entsprechenden Rechten (Userprofil Administrator) Zugriff. Der Anwender kann hier die Stammdaten Modell, User, Rolle und Berechtigung sowohl anlegen, editieren als auch löschen.

Die zweite Ansicht ist der Modeller. Dies ist der eigentliche Client, auf den die Anwender zugreifen, denn in diesem findet das Modellieren der BPMNs statt.

## Datenbank

Aus Kompatibilitätsgründen mit Camunda, welches im originalen Zustand ebenfalls an Tabellen zur Speicherung von Daten bedient, wurde PostgreSQL als relationale Datenbank aufgesetzt. Die Datenbank liegt auf dem Server der Hochschule Landshut und kann entweder direkt über das Hochschulnetz oder über VPN erreicht werden. Eine Anleitung für den VPN Zugang ist in Kapitel 3.4 zu finden.

Der Aufbau der Datenbank beinhaltet das folgende Schema:

Bei den Tabellen Model, User, Userprofile, Role, Permission und Partialmodel handelt es sich um die Stammdaten. Die Bewegungsdaten werden in der Tabelle Session abgebildet.

Session steht für sich allein und dient überwiegend technischen Gründen. Session wird zur Authentifizierung beim Login eines Users aufgerufen und erzeugt für jeden User, der sich einloggen möchte eine gültige Session ID.

Userprofil definiert verschiedene Profile mit unterschiedlichen Rechten. Die Rechte werden als bitfield gespeichert, was bedeutet, dass ein Profil entweder Rechte besitzt oder nicht. Die Tabelle Userprofil legt damit die Zugriffsrechte auf die Administrationsseite fest.

User stellt die Anwender des ADAMO dar. Diese Tabelle besitzt neben seinen Attributen zudem den Fremdschlüssel der ID der Userprofil Tabelle. Einem User muss über die Administrationsseite ein entsprechendes Profil zugewiesen werden, sodass dieser entweder als Administrator fungiert oder nur auf die Modellierung zugreifen kann. Ein User kann nur ein Profil innehaben. Dies wird über eine 1 zu 1 Beziehung abgebildet.

Model beinhaltet die Modelle, die von den Anwendern im ADAMO erstellt werden. Ein Modell kann entweder Teilmodelle enthalten oder als ein Teilmodell fungieren. Die Tabelle Partialmodel enthält daher als Fremdschlüssel die Kombination aus der ID und der Version der Model Tabelle. Da mehrere Modelle ein oder mehrere Teilmodelle besitzen können und selbst in anderen Modellen als Teilmodell fungieren können wird hier eine n zu m Verknüpfungen zwischen der Model Tabelle und der Partialmodel Tabelle abgebildet.

Role definiert verschiedene Rollen mit unterschiedlichen Rechten. Die Rechte unterscheiden sich hinsichtlich lesenden Rechten („read“), schreibenden Rechten („write“) oder Administrationsrechten („admin“). Eine Rolle wird mit den entsprechenden Berechtigungen gespeichert.

Permission legt fest, mit welcher Rolle, und damit mit welchen Rechten ein User auf ein Modell zugreifen darf. Diese Tabelle enthält damit als Fremdschlüssel die IDs der Tabellen User, Model und Role in einer n zu m Beziehung.

Abbildung 1 veranschaulicht das voran erklärte Datenbankschema mit seinen Beziehungen:

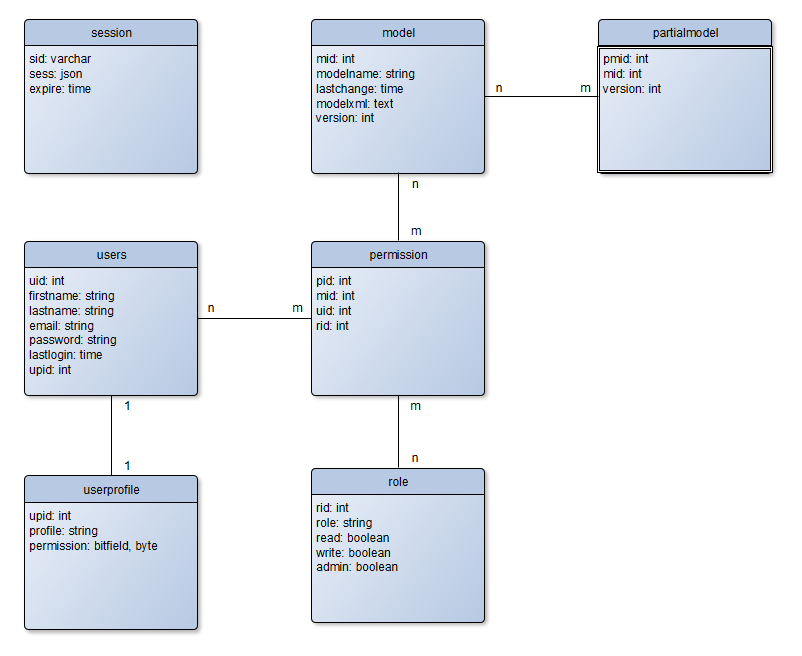


Abbildung 1: Datenbanktabellen und ihre Beziehungen zueinander

## MQTT

Für das Studienprojekt wurde ein Mosca Server für das Austauschen von Nachrichtenprotokollen aufgesetzt, welcher das MQTT Protokoll implementiert und so den Usern unter anderem ermöglicht für bestimmte Modelle zu „subscriben“. Dadurch können mehrere User gleichzeitig an einem BPMN Modell arbeiten. Die Änderungen werden live an allen Usern übertragen, sodass diese immer über den aktuellsten Stand der Modelle verfügen.

Mosca wurde in diesem Projekt in der Version 2.7.0 und MQTT in der Version 2.16.0 verwendet.

Abbildung 2 stellt das Publish/Subscriber Modell bei MQTT grafisch dar.

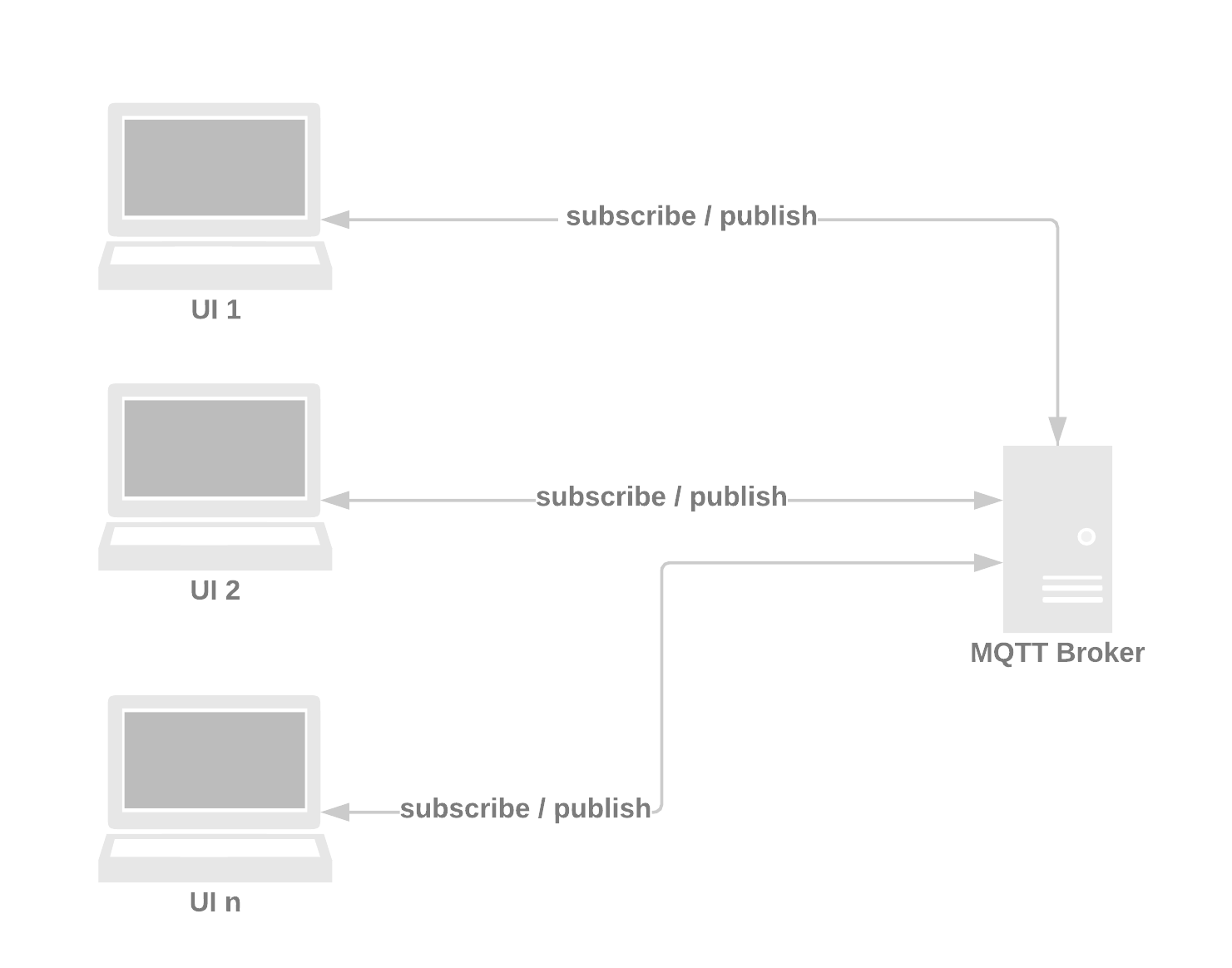


Abbildung 2: Publish/Subscriber Modell bei MQTT

Die Mosca Server Datei für MQTT (mqttserver.js) liegt im Ordner „API“. Damit das Subscriben und das Laden der neuesten Versionen der Modelle funktioniert, muss vor jedem Start des Programms zuerst über der Konsole zum API Pfad navigiert werden. Dies geschieht über den Befehl „cd API“. Es wird nun der API Pfad angezeigt. Abschließend muss über den Befehl „node mqttserver.js“ der Server gestartet werden.

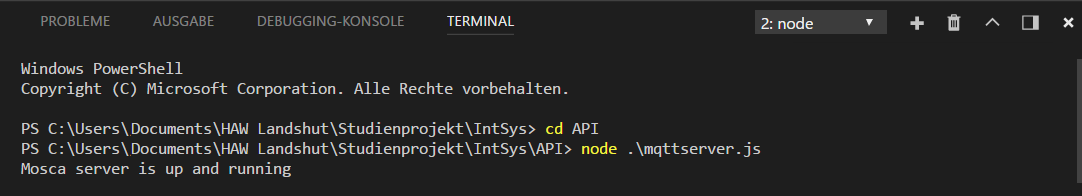


Abbildung 3: Starten des Mosca-Servers über die Konsole

## Express.js

Express ist ein einfaches und flexibles serverseitiges Framework für node.js. Express erweitert node.js um zahlreiche leistungsfähige Funktionen und Werkzeuge, sodass moderne Webanwendungen und mobile Anwendungen bereitgestellt werden können.

In den Express.js-Server Dateien sind die HTTP-Endpoints definiert. Die zentrale Datei ist die express.js, mit welcher der Server gestartet wird. Die express.js-Datei mapped die Requests anhand der URL auf die entsprechenden Servlets user, permission, model, partmodel, profile und role. Diese wiederrum beinhalten jede für sich die CRUD Endpoints der jeweiligen Datenbanktabelle. Das bedeutet, dass bspw. in der user.js definiert ist, wie serverseitig ein User anzulegen, zu bearbeiten und zu löschen ist.

Damit der Webserver gestartet werden kann, muss zunächst wieder in den API Pfad navigiert werden. Der Express.js Server wird über den Befehl „node express.js“ gestartet.

Express.js wurde im Projekt in der Version 4.16.2 inkludiert.

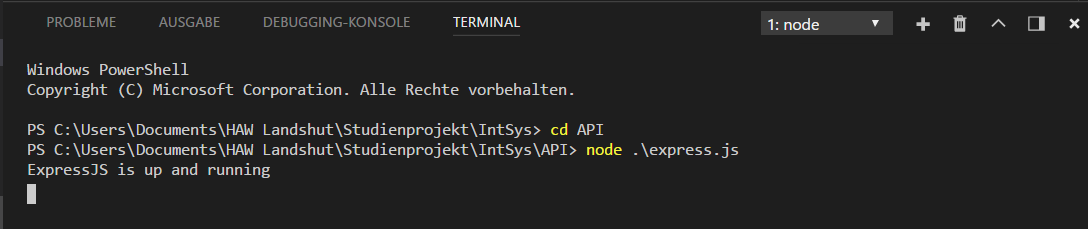


Abbildung 4: Starten des express.js-Servers über die Konsole

## Angular Frontend

Da Angular derzeit als Standard für dynamische Webentwicklung gesehen wird, wurde dieses für die Implementierung der Funktionen der Oberfläche und damit der Präsentationsschicht verwendet. Von der Version her betrachtet kamen Angular 2 und das zugehörige Webpack 1.0.0, welche mit der bpmn.io zu Projektstart kompatibel waren.

Abgesehen von der Oberflächenlogik wird eine gestalterische Komponente benötigt, wofür Bootstrap in Verbindung mit den Glyphicons in den eingebunden wurde. Die Icons befinden sich in der Palette wie später in Kapitel 6.1.2.5 gezeigt und können unter <https://getbootstrap.com/docs/3.3/components/> für die aktuelle im Projekt verwendete Bootstrap Version 3.3.7 ausgewählt werden.

# Installationsanleitung

## Git bzw. GitKraken

Zur Versionierung von Code-Änderungen wird ein Git-Repository verwendet. Um dieses benutzerfreundlich zu verwalten kann GitKraken verwendet werden, zu dem ein Download unter <https://www.gitkraken.com/> möglich ist.

Im nächsten Schritt muss das Repository geklont werden. Dieses befindet sich auf GitHub unter <https://github.com/HAWMobileSystems/IntSys>, wofür zum Klonen folgender Link verwendet wird: [git@github.com:HAWMobileSystems/IntSys.git](mailto:git@github.com:HAWMobileSystems/IntSys.git)

GitKraken nach der Installation starten und die oben genannte URL unter

„File“ 🡪 „Clone Repo“

einfügen. Im nachfolgenden Beispiel sollte das Ziel durch einen lokal sinnvollen, leicht aufzufindenden Pfad ersetzt werden.

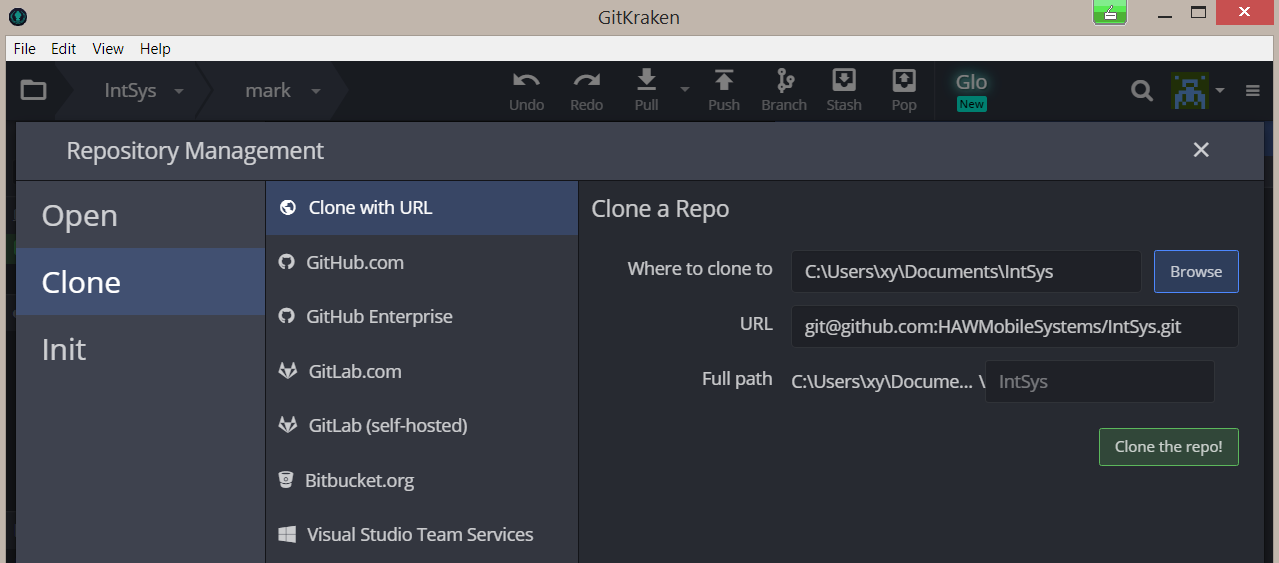


Abbildung 5: Klonen eines Repositories mittels GitKraken

## Visual Studio Code

Außerdem wird für weitere Implementierungen eine Entwicklungsumgebung benötigt, wofür Visual Studio Code eingesetzt werden kann, da dies ebenso Git unterstützt. Ein weiterer Vorteil dieses Tools ist das leichte Debuggen von Webanwendungen. Der Installer hierfür ist unter <https://code.visualstudio.com> zu finden.

Als zusätzliche Komponenten werden vom Hersteller selbst Git, Node.js und TypeScript empfohlen, dies ist jedoch mit den dependencies im Projekt (siehe Kapitel 6.1.5) bzw. dem im Nachgang erläuterten express.js bereits abgedeckt. Als empfehlenswerte Erweiterungen bezüglich Code-Style zeigen sich jedoch die \*lint Komponenten. Diese können wie im nachfolgenden Screenshot gezeigt mittels des gelb markierten Buttons gefiltert, selektiert und installiert werden.

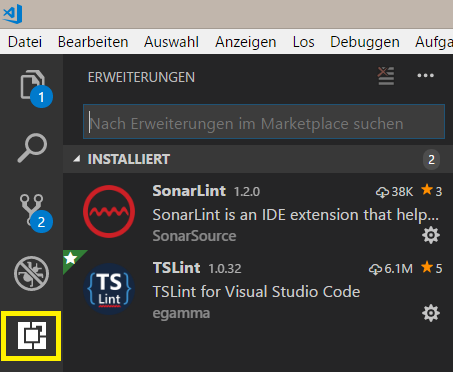


Abbildung 6: Installation von zusätzlichen Komponenten

## Angular und express.js

Voraussetzung, um das Setup des express.js-Servers und des Angular-Frontends durchzuführen, ist die Installation von Node.js. Dieses kann über den Link <https://nodejs.org/en/download/> je nach Betriebssystem heruntergeladen und installiert werden.

Nachdem das Repository, wie in Kapitel 3.1 beschrieben, erfolgreich geklont wurde muss nur noch ein npm install in den Foldern API und angular2 durchgeführt werden.

Hierzu eine Konsole öffnen und in den API Ordner wechseln. Im Anschluss den Befehl npm install eingeben.

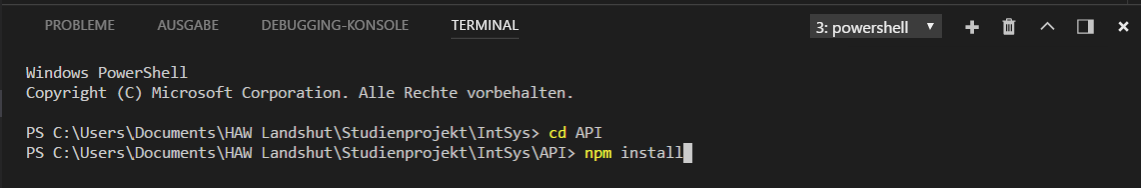


Abbildung 7: npm install im API Ordner auf dem express.js-Server

Um denselben Schritt im Frontend durchzuführen eine zweite Konsole öffnen (s. grüne Umrandung) und ins angular2-Verzeichnis wechseln. Auch dort nochmal den Befehl den npm install ausführen.

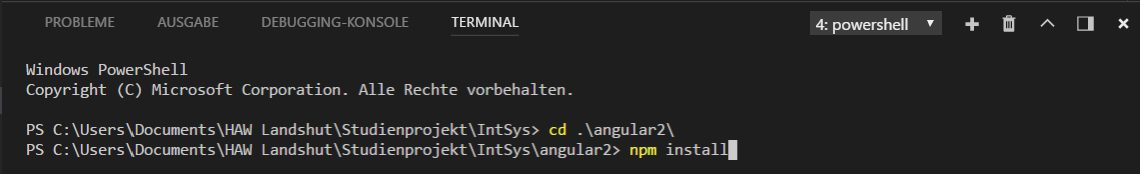


Abbildung 8: npm install im angular2-Ordner im Frontend

## Open VPN

Da sich die Datenbank auf einem Server im Hochschulnetz befindet, wird eine VPN-Verbindung benötigt. Die Hochschule stellt hierfür eine Open VPN GUI bereit, welcher unter folgendem Link für das gewünschte Betriebssystem heruntergeladen werden kann:  
<https://www.haw-landshut.de/hochschule/zentrale-services/service-it/download.html>

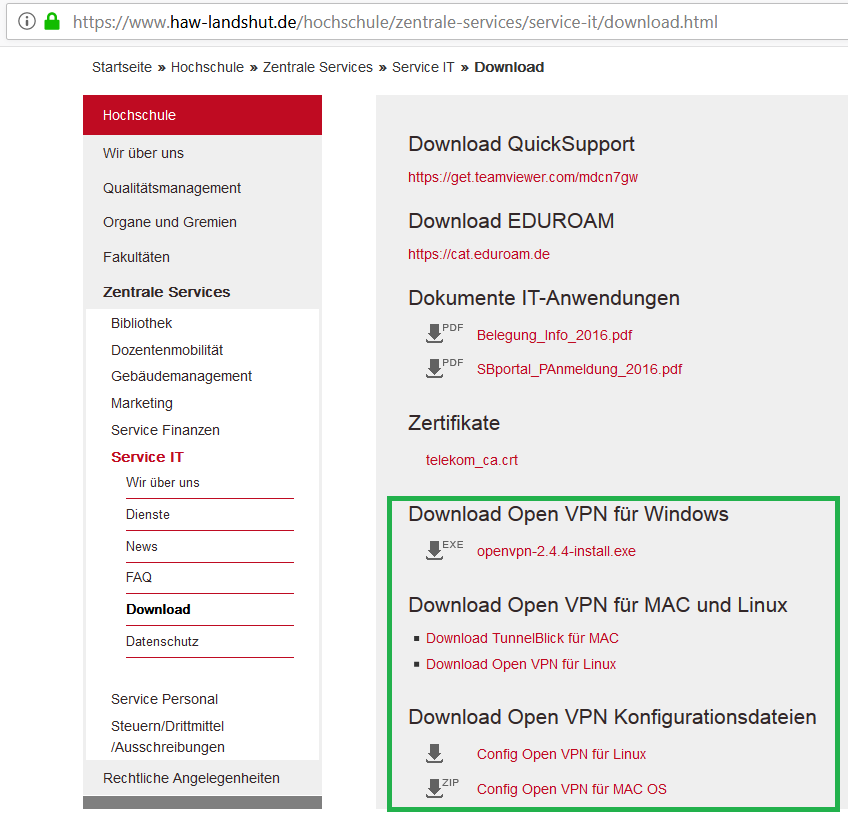


Abbildung 9: Download-Möglichkeiten von Open VPN der HAW Landshut

## PostgreSQL und pgAdmin (optional)

Der Download von PostgreSQL empfiehlt sich in erster Linie für alle, die eine lokale Datenbank aufsetzen möchten. Ein Download von Postgres ist unter <https://www.postgresql.org/download/> möglich. Postgres ist dann auf dem entsprechenden Betriebssystem zu installieren.

Wenn die Installation erfolgreich abgeschlossen ist empfiehlt sich zudem ein Postgres-Browser, der dem Benutzer die Bedienung deutlich erleichtert. Als Postgres-Browser haben wir uns für pgAdmin 4 entschieden. Dieser kann über <https://www.pgadmin.org/download/pgadmin-4-windows/> heruntergeladen und installiert werden. pgAdmin steht nach erfolgreicher Installation sofort zur Verfügung.

Weitere Schritte für die Erstellung einer lokalen Datenbank sind in Kapitel 4.1 Anlage einer lokalen Datenbank zu finden.

# Lokale und Server-Datenbank

## 4.1 Anlage einer lokalen Datenbank

Dieses Kapitel beschreibt die Anlage einer lokalen Datenbank in pgAdmin.

Zunächst muss ein neuer Server angelegt werden, auf dem die Datenbank laufen soll. Hierzu nach dem Start von pgAdmin einen Rechtsklick auf Servers ausführen und auf „Create > Server“ navigieren. Daraufhin öffnet sich ein neues Fenster, in dem die folgenden notwendigen Informationen hinterlegt werden müssen:

* Tab General: Name: ipim
* Tab Connection: Host name/adress: localhost

Port: 5432

Maintenance database: postgres

Username: postgres

Password: 12341234

Save password: Checkbox anhaken

Nachdem der Server entsprechend erzeugt wurde, beinhaltet er per Default die Datenbank „postgres“. Die Datenbank des ADAMO kann entweder über die GUI des pgAdmins oder über Query Tools erzeugt werden. Da die Anlage mit Query Tools schneller geht, wird in der Dokumentation nur dieser Weg beschrieben. Hierfür durch einen Rechtklick auf die Datenbank „postgres“ Query Tool aufrufen. In dem sich neu öffnenden Fenster auf der rechten Seite kann mit den nachfolgenden Befehlen die Datenbank lokal erzeugt werden. (Alternativ kann eine Kopie des bestehenden Servers verwendet werden.

### -- Database: ipim

CREATE DATABASE ipim

WITH

OWNER = postgres

ENCODING = 'UTF8'

LC\_COLLATE = 'German\_Germany.1252'

LC\_CTYPE = 'German\_Germany.1252'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1;

### -- Table: public.session

CREATE TABLE session (

sid character varying NOT NULL,

sess json NOT NULL,

expire timestamp(6) without time zone NOT NULL

CONSTRAINT session\_pkey PRIMARY KEY (sid)

);

ALTER TABLE session OWNER TO postgres;

### -- Table: public.users

CREATE TABLE public.users

(

uid serial NOT NULL,

firstname text NOT NULL,

lastname text NOT NULL,

email text NOT NULL,

password text NOT NULL,

lastlogin timestamp without time zone NOT NULL,

upid serial,

CONSTRAINT users\_pkey PRIMARY KEY (uid),

CONSTRAINT users\_email\_key UNIQUE (email),

CONSTRAINT upid FOREIGN KEY (upid)

REFERENCES public.userprofile (upid) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (OIDS = FALSE)

TABLESPACE pg\_default;

ALTER TABLE public.users OWNER to postgres;

### -- Triggerfunction: set\_lastlogin

CREATE FUNCTION public. set\_lastlogin()

RETURNS trigger

LANGUAGE 'plpgsql'

NOT LEAKPROOF

AS $BODY$

BEGIN

NEW.lastlogin := NOW();

RETURN NEW;

END;

$BODY$;

ALTER FUNCTION public. set\_lastlogin() OWNER TO postgres;

### -- Trigger: set\_lastlogin

CREATE TRIGGER set\_lastlogin

BEFORE INSERT OR UPDATE

ON public.users

FOR EACH ROW

EXECUTE PROCEDURE public.set\_lastlogin();

### -- Table: public.model

CREATE TABLE public.model

(

mid serial NOT NULL,

modelname text NOT NULL,

lastchange timestamp without time zone DEFAULT now(),

modelxml text,

version bigint NOT NULL,

CONSTRAINT model\_pkey PRIMARY KEY (mid, version)

)

WITH (OIDS = FALSE)

TABLESPACE pg\_default;

ALTER TABLE public.model OWNER to postgres;

### -- Triggerfunction: set\_lastchange()

CREATE FUNCTION public.set\_lastchange()

RETURNS trigger

LANGUAGE 'plpgsql'

COST 100

VOLATILE NOT LEAKPROOF

AS $BODY$

BEGIN

NEW.lastchange := NOW();

RETURN NEW;

END;

$BODY$;

ALTER FUNCTION public.set\_lastchange() OWNER TO postgres;

### -- Trigger: set\_lastchange

CREATE TRIGGER set\_lastchange

BEFORE INSERT OR UPDATE

ON public.model

FOR EACH ROW

EXECUTE PROCEDURE public.set\_lastchange();

### -- Table: public.partialmodel

CREATE TABLE public.partialmodel

(

pmid serial NOT NULL,

mid serial NOT NULL,

version bigint,

CONSTRAINT mid FOREIGN KEY (mid, version)

REFERENCES public.model (mid, version) MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE NO ACTION)

WITH (OIDS = FALSE)

TABLESPACE pg\_default;

ALTER TABLE public.partialmodel OWNER to postgres;

### -- Table: public.permission

CREATE TABLE public.permission

(

pid serial,

mid serial,

uid serial,

rid serial,

CONSTRAINT permission\_pkey PRIMARY KEY (pid),

CONSTRAINT permission\_uid\_mid\_key UNIQUE (uid, mid),

CONSTRAINT rid FOREIGN KEY (rid)

REFERENCES public.role (rid) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT uid FOREIGN KEY (uid)

REFERENCES public.users (uid) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION)

WITH (OIDS = FALSE)

TABLESPACE pg\_default;

ALTER TABLE public.permission OWNER to postgres;

### -- Table: public.role

CREATE TABLE public.role

(

rid serial NOT NULL,

role text NOT NULL,

read boolean,

write boolean,

admin boolean,

CONSTRAINT role\_pkey PRIMARY KEY (rid)

)

WITH (OIDS = FALSE)

TABLESPACE pg\_default;

ALTER TABLE public.role OWNER to postgres;

### -- Table: public.userprofile

CREATE TABLE public.userprofile

(

upid serial NOT NULL,

profile text NOT NULL,

permission bit(1) NOT NULL,

CONSTRAINT userprofile\_pkey PRIMARY KEY (upid)

)

WITH (OIDS = FALSE)

TABLESPACE pg\_default;

ALTER TABLE public.userprofile OWNER to postgres;

## 4.2 Wechsel zwischen lokaler und Server-Datenbank

Wenn eine lokale Datenbank erstellt wurde, kann selbstverständlich auch zu dieser eine Verbindung aufgebaut werden. Hierzu ist zunächst die Verbindung zum Hochschulserver zu beenden und die lokale Verbindung anzugeben.

Die database.js Datei öffnen und die folgenden Daten durch die daten der lokalen Datenbank (s. Kapitel 4.1 Anlage einer lokalen Datenbank) ersetzen:

|  |  |
| --- | --- |
|  | Lokale Datenbank |
| host | localhost |
| port | 5432 |
| database | ipim |
| user | postgres |
| password | 12341234 |

Tabelle 1: Verbindungsdaten der lokalen Datenbank

# Starten des ADAMO für Entwickler

Nachfolgend beschriebene Schritte müssen bei jedem Start des Projekts durchgeführt werden.

1. Entweder die lokale Datenbank erstellt haben oder mit VPN zur Server-Datenbank verbinden
2. Visual Studio Code starten
3. Express.js-Server starten

* Falls das Terminal nicht automatisch geöffnet ist, dieses über den Menüpunkt „Anzeigen“ -> „Integriertes Terminal“ öffnen
* Mit „cd API“ in den API Ordner wechseln
* „node express.js“ ausführen, um startup, routing und andere Basisfunktionen zu starten. Falls eine Fehlermeldung bezüglich Timeout erscheint, sollte die VPN-Verbindung geprüft werden

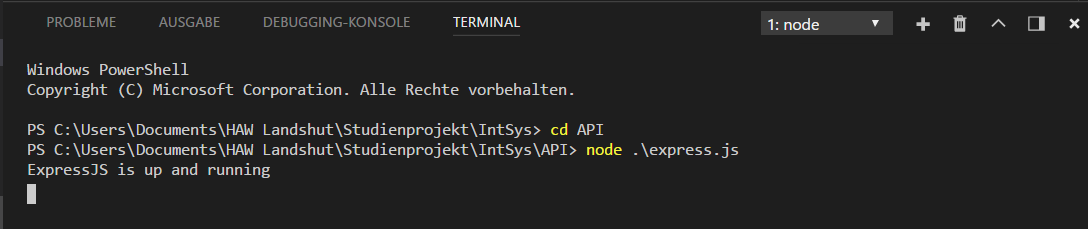


Abbildung 10: express.js starten

1. MQTT starten

* Um die Kommunikation zwischen mehreren Endgeräten zu testen, muss der MQTT gestartet werden. Hierzu eine neue Konsole öffnen und mit „cd API“ abermals in den API-Ordner wechseln
* „node mqttserver.js“ ausführen, um den MQTT starten.

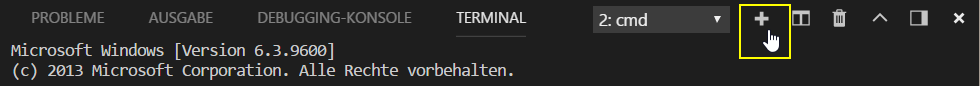


Abbildung 11: Öffnen einer neuen Konsole

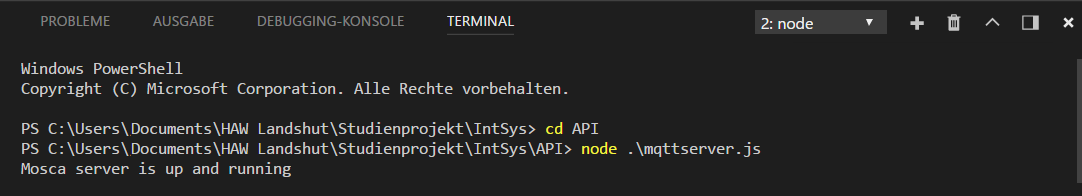


Abbildung 12: Starten des MQTT-Servers

1. Frontend starten

* Eine dritte Konsole öffnen und mit „cd angular2“ ins Angular-Verzeichnis wechseln
* Den express.js bzw. node.js Paketmanager mit „npm start“ starten. Im Idealfall erscheint am Ende die Meldung „webpack: Compiled successfully.“.   
  Andernfalls werden bei einem nicht complierbarem Code hier die Fehlerstellen angezeigt. Wenn trotz genauerer Betrachtung des Codings keine Fehlerquellen auffindbar sind, hilft ein erneutes „npm install“. Dieses ist beispielsweise nach Änderungen an den Dependencies notwendig

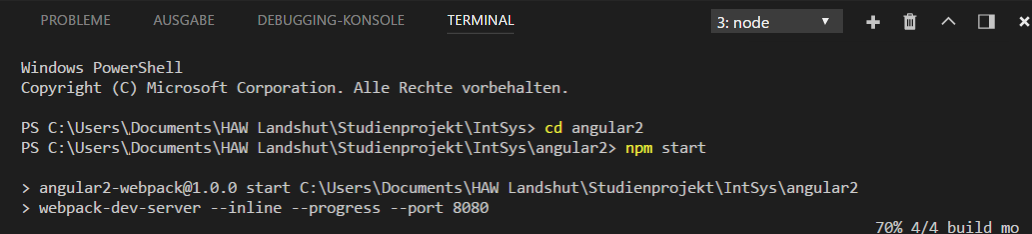


Abbildung 13: Starten des Node-Paketmanagers

1. Den Browser öffnen und die localhost:8080 als URL eingeben
2. Login-Daten für Testzwecke:

User: user@demo.com

Passwort: 12341234

# Aufbau der Anwendung im VS Code

Die Dateistruktur in VS Code ist in zwei relevante Ordner aufgeteilt: angular2 und API. Im angular2-Ordner befinden sich alle für die GUI relevanten Aspekte, wohingegen im API-Ordner alle für die Kommunikation in der Schnittstelle und Datenbank relevanten Dateien abgelegt sind. Welche dies im Detail sind und wie diese mit der Oberfläche zusammenhängen wird nun erläutert.

## Aufbau der GUI

### Login

Die Login-Seite ist unter der Sub-URL „front-page“ (http://localhost:8080/front-page) erreichbar, die Logik dahinter befindet sich in der Komponente „front-page.component.ts“ und die Oberflächengestaltung in „front-page.component.html“ sowie „front-page.component.css“. Wie die Farbgestaltung geändert werden kann ist in Kapitel 6.1.4 beschrieben.

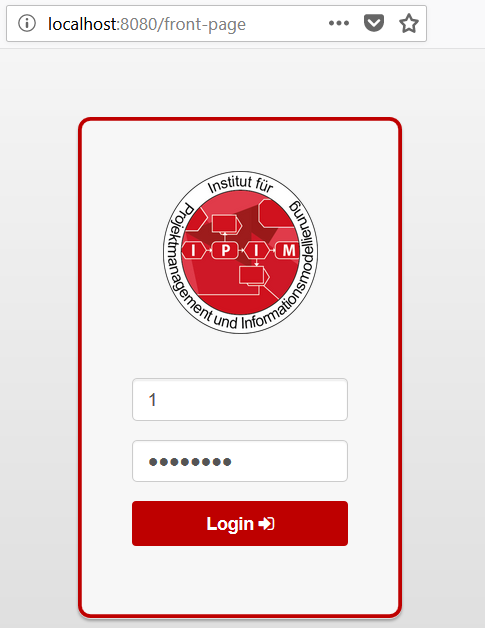
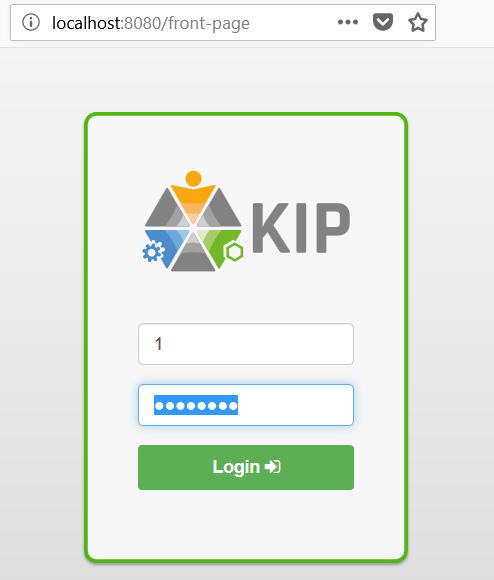


Abbildung 14: Login Page im KIP-Design vs. Login Page im IPIM-Design

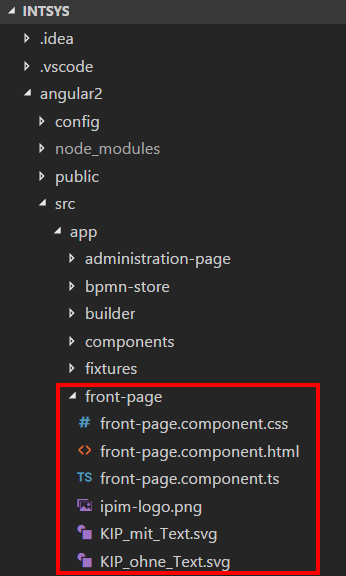


Abbildung 15: Front Page Struktur in VS Code

### Modeller

Nach dem Login wird der User auf die modellerPage geleitet, wo er ein Modell auswählen oder anlegen und im Anschluss bearbeiten kann. Verfügbar ist dieser unter der Sub-URL „modeler“ (http://localhost:8080/modeler).

#### modellerPage

Die modellerPage ist eine Übersichtsseite für alle existierenden Modelle, auf die ein Anwender lesenden oder schreibenden Zugriff hat. Der User kann sich hier eines der Modelle in der gewünschten Version auswählen, welches dann über „Load from DB“ als BPMN geladen wird. Zudem kann er über „Import to DB“ das selektierte Modell lokal abspeichern. Über „New“ kann der User ein neues Modell erstellen.

Zudem findet sich in dieser Ansicht auch eine Übersicht aller Modelle, die sich in den letzten sieben Tagen geändert haben.

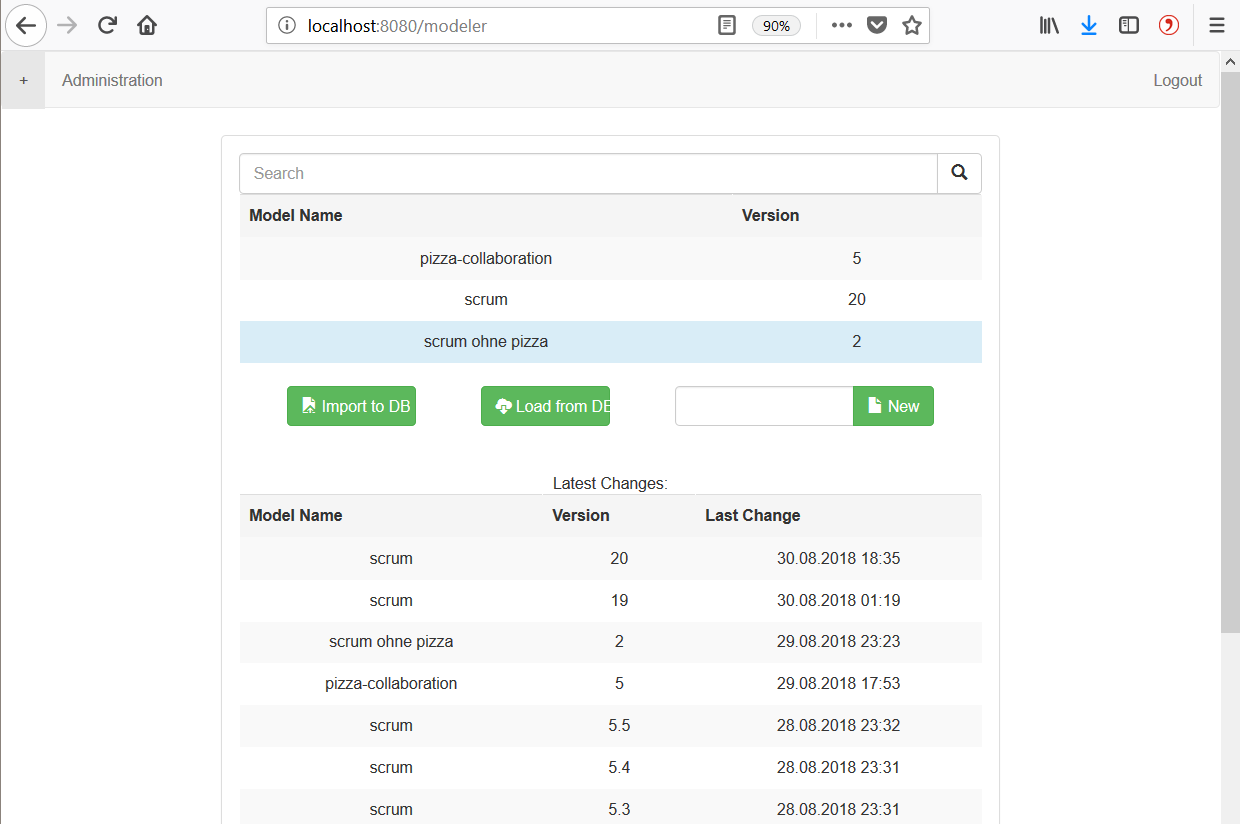


Abbildung 16: Einstiegsseite zum Modeller: modellerPage

Das zugehörige Coding findet sich im Paket modellerPage.

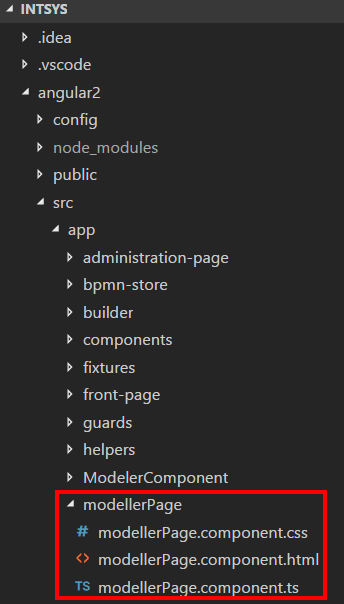


Abbildung 17: modellerPage Struktur in VS Code

#### Model

Diese Ansicht ist in mehrere Abschnitte aufgeteilt. Einerseits werden in der Navigation Bar, welche in Kapitel 6.1.2.4 beschrieben ist, die geöffneten Modelle und Mitwirkende angezeigt, andererseits splittet sich der eigentliche Modeller nochmals in eine Palette mit den in diesem Projekt implementierten Funktionen und den Modellierungswerkzeugen, siehe Kapitel 6.1.2.5, sowie dem eigentlichen Modell und einer Panel für die Einstellungen.

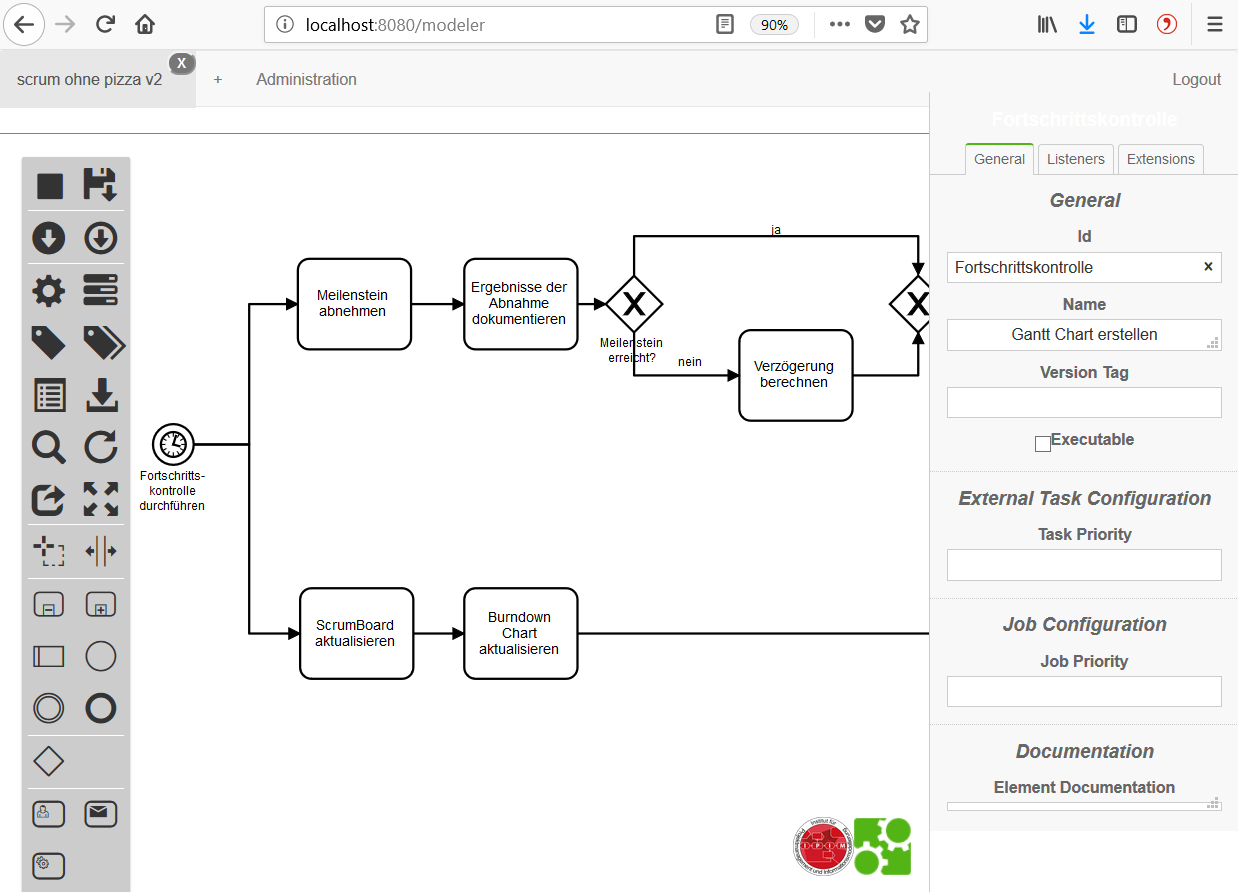


Abbildung 18: Modeller mit Properties Panel

Das eigentliche Modell ist im File „model.ts“ klassifiziert.

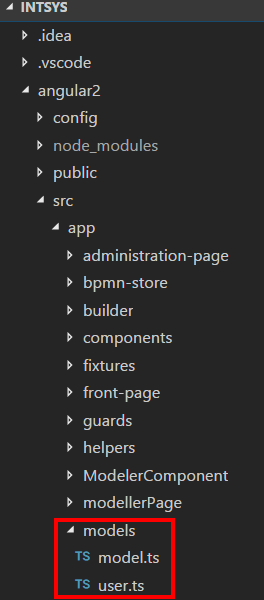


Abbildung 19: Definition von Model und User

#### Versionierung

Sobald ein Modell in der modellerPage ausgewählt wurde, werden, sofern es weitere Versionen des Modells gibt, die vorangegangenen Versionen angezeigt. Dabei ist die Logik zur Versionierung folgendermaßen:

* Wird ein Modell geladen, verändert und anschließend gespeichert, wird die Versionsnummer um eins hochgezählt (Beispiel: 1=>2)
* Wird ein älteres Modell einer Subversion geladen, verändert und anschließend gespeichert, wir die Subversion dieser Ebene um eins hochgezählt (Beispiel: 1.1 => 1.2)
* Wird ein älteres Modell geladen, verändert und anschließend gespeichert, es existiert jedoch noch keine Subversion dazu (bspw. liegt ein Modell in Version 1 und 2 vor), wird zum älteren Modell eine Subversion angelegt (Beispiel: 1 => 1.1)
* Durch den Datentyp Big Integer ist die höchste Version 65535
* Die maximale Verschachtelung ist bis Ebene vier möglich
* Die höchste Versionsnummer mit maximaler Verschachtelung beträgt:   
  65535. 65535. 65535. 65535

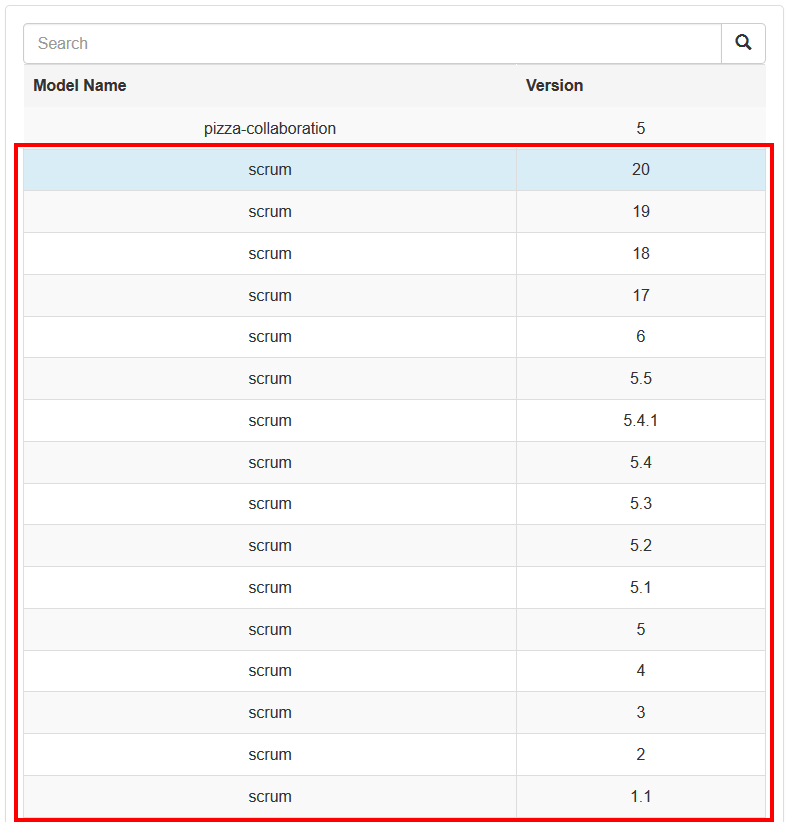


Abbildung 20: Versionierung von Modellen anhand eines scrum Beispiels

Die Logik zu der Berechnung der Versionsnummer findet sich in version.pipe.ts eine genauere Erläuterung zu dieser kann den dortigen Kommentaren entnommen werden.

Außerdem wird die Version beim SaveModal aufgerufen und beim Speichern eines Modals entsprechend hochgezählt. Wie das SaveModal aufgerufen werden kann ist in Kapitel 6.1.2.6 aufgezeigt.

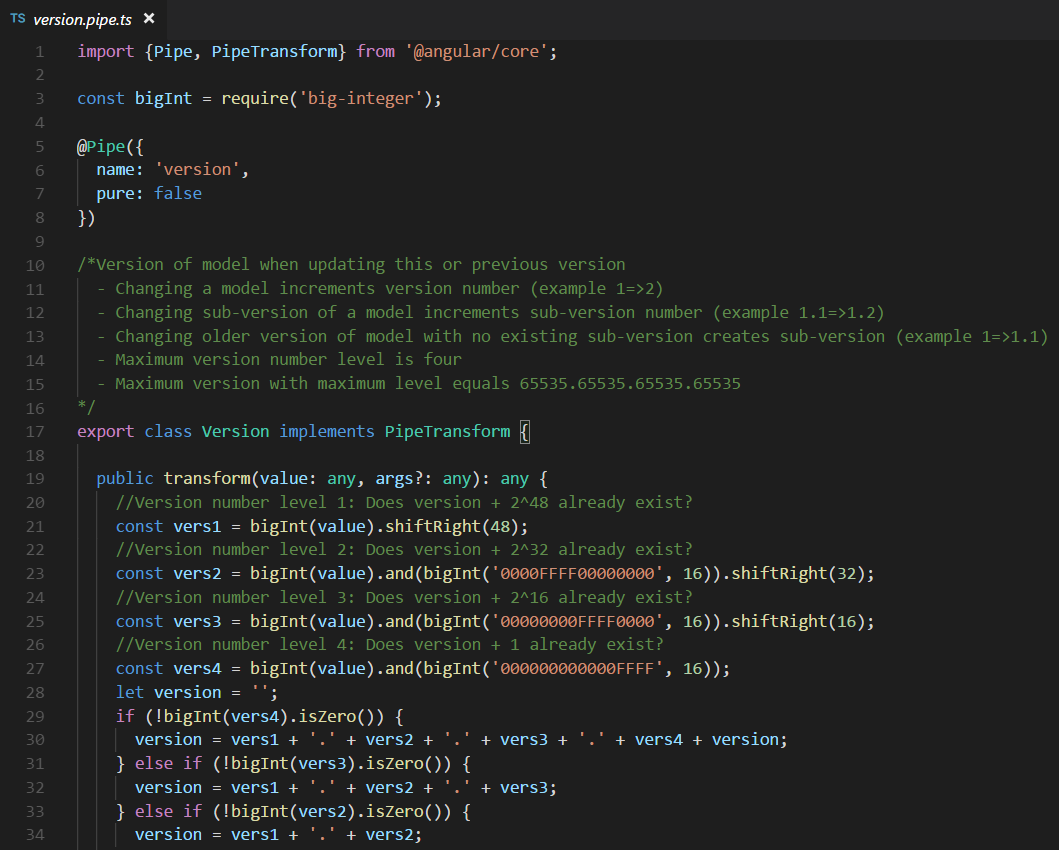


Abbildung 21: Logik zur Versionierung von Modellen

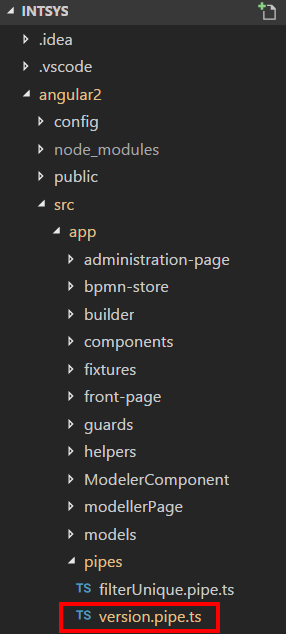


Abbildung 22: File zur Versionierung in der Dateistruktur

#### Navigation Bar

In der Navigation Bar wird sowohl die Version eines Modells als auch die derzeit an dem Modell arbeitenden User angezeigt.

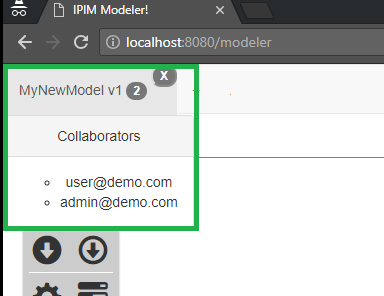


Abbildung 23: Anzeige eines Modells in der Navigation Bar

Der Aufbau zu dieser findet sich beim Modeller unter modellerPage.component.html und für die Administration Page unter administration-page.component.html und besteht im Wesentlichen aus dem Modellnamen, der Versionsnummer und der Anzahl der Mitmodellierenden sowie einem Schließen-Button in den einzelnen Tabs zu den Modellen. Außerdem kann mit dem „+“ ein neuer Tab erzeugt werden, in dem ein Modell geöffnet wird. Der Absprung zur Administration ist nur sichtbar, wenn der eingeloggte User die benötigte Berechtigung und dazu das passende Profil besitzt.

#### Modeller-Palette

Zu linker Hand findet sich im Modeller eine Palette mit weiteren Funktionen zur Bearbeitung der gezeichneten Prozesse. Hierbei kann die Anzeige dieser je nach Bedarf zwischen einer und zwei Spalten gewechselt werden. Die einzelnen Funktionen lassen sich bei Unsicherheit durch den Tooltip bei einem Mouseover herausfinden.

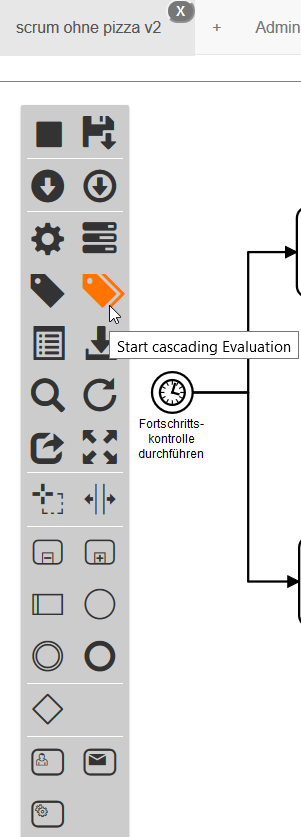


Abbildung 24: Palette mit Icons für weitere Funktionen

Ist gewünscht, die Palette zu erweitern oder Icons auszutauschen, werden die beiden Dateien bpmn-store.service.ts und palette.ts benötigt. In ersterer sind insbesondere die für zusätzliche Funktionen benötigten Icons, in zweiterer die bereits von Camunda zur Verfügung gestellten und für unsere Zwecke angepassten Icons hinterlegt. Die Icons werden einem Kommando zugeordnet und eine Aktion bei der Auswahl hinterlegt, wobei das Kommando in der commandstore.service.ts definiert ist. Hier werden Glyphicons verwendet, welche wie in Kapitel 2.4 erwähnt unter <https://getbootstrap.com/docs/3.3/components/> ausgewählt werden können.

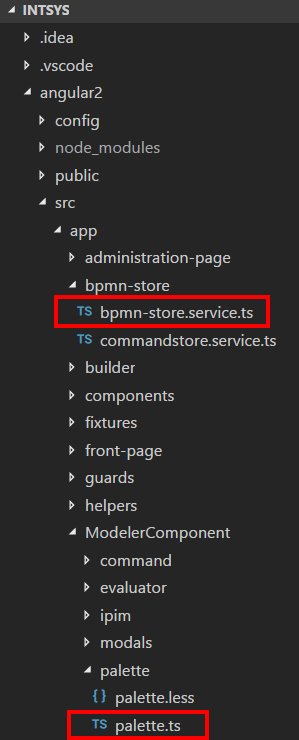


Abbildung 25: Dateien zur Palettenverwaltung

#### Modale

Zu vielen Icons in der Palette öffnet sich ein Modal, welches dem User weitere Optionen bietet. Im Detail sind dies das Speichern von Modellen (SaveModal), das Setzen von Variablen (VariableModal), das Setzen von Termen (TermModal), das Evaluieren von Prozessen (InputModal), das kaskadierende Evaluieren von Prozessen (evaluatorModal) und das Setzen von Subprozess IDs (SubProcessModal).

Zwei Beispiele zeigen die folgenden Screenshots:

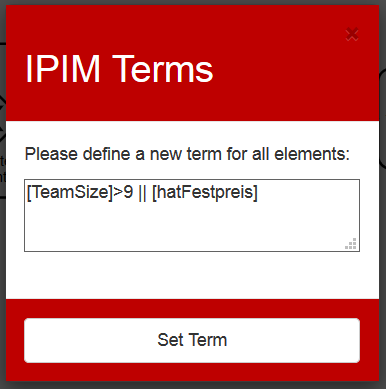


Abbildung 26: Term Modal

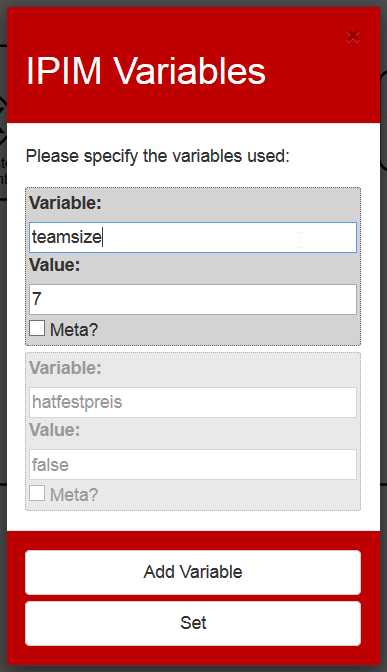


Abbildung 27: Variablen Modal

Die übergreifende Gestaltung aller Modale ist in der Modal.css und die Methodenaufrufe in den jeweiligen \*.ts Files enthalten. Das Input und auch das Variablen Modal besitzen jeweils eine Komponente, welche für das dynamische Auslesen der Werte und den Aufbau des Modals verantwortlich sind (InputComponent und VariablesComponent).

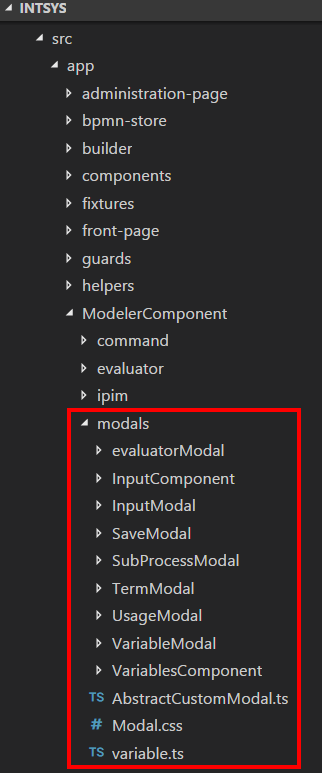


Abbildung 28: Dateistruktur zu den Modalen

Der Komponentenaufruf wird über den selector der Komponente getriggert. Nachfolgend ein Beispiel für das Variablen Modal:

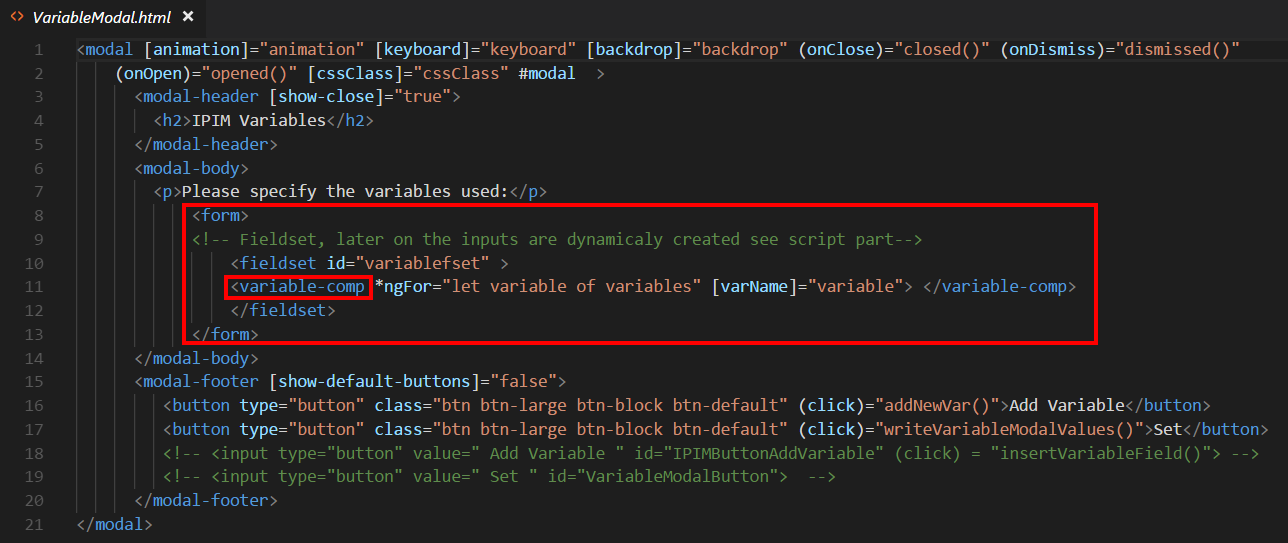


Abbildung 29: Aufruf der Komponente aus dem Variablen Modal heraus

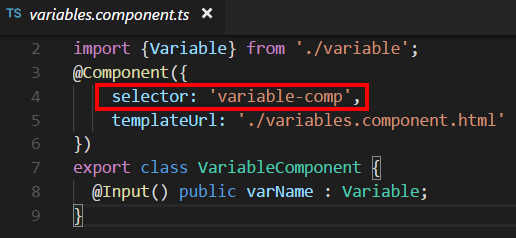


Abbildung 30: Komponente zum Variablen Modal enthält den zugehörigen selector

### Modeller Funktionen

Folgende zusätzliche Funktionen wurden im Projekt implementiert und können über die Icons der Palette im Modeller aufgerufen werden. Die Bedienung der einzelnen Elemente kann dem Benutzerhandbuch entnommen werden. Für das Coding ist der zentrale Ausgangspunkt die modeler.component.ts Datei, welche alle zugehörigen Modale und ähnliches importiert und selbst auch viel weitere Logik enthält. Im nachfolgenden werden nun die einzelnen Funktionen aufgezählt und die relevanten Codingsstellen zugeordnet.

* Export to BPMN

Diese Funktion ist in modeler.component.ts implementiert:

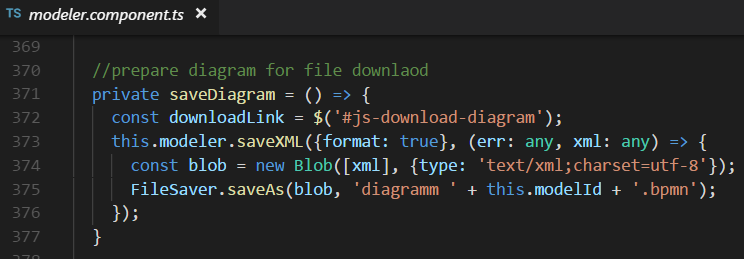


Abbildung 31: Download eines Modells in bpmn-Format

* Export to SVG

Diese Funktion ist in modeler.component.ts implementiert:

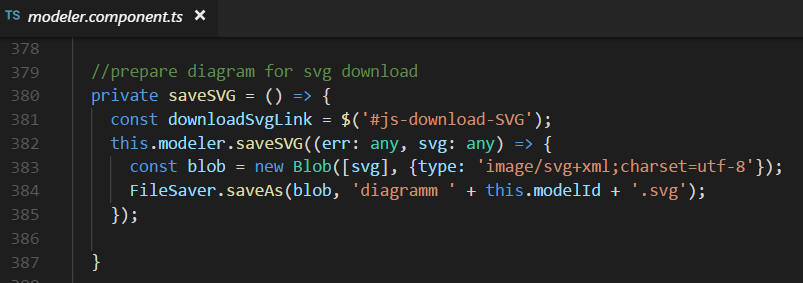


Abbildung 32: Download eines Modells im svg-Format

* Set Variables

Diese Funktion ist im VariableModal implementiert (siehe Kapitel 6.1.2.6:

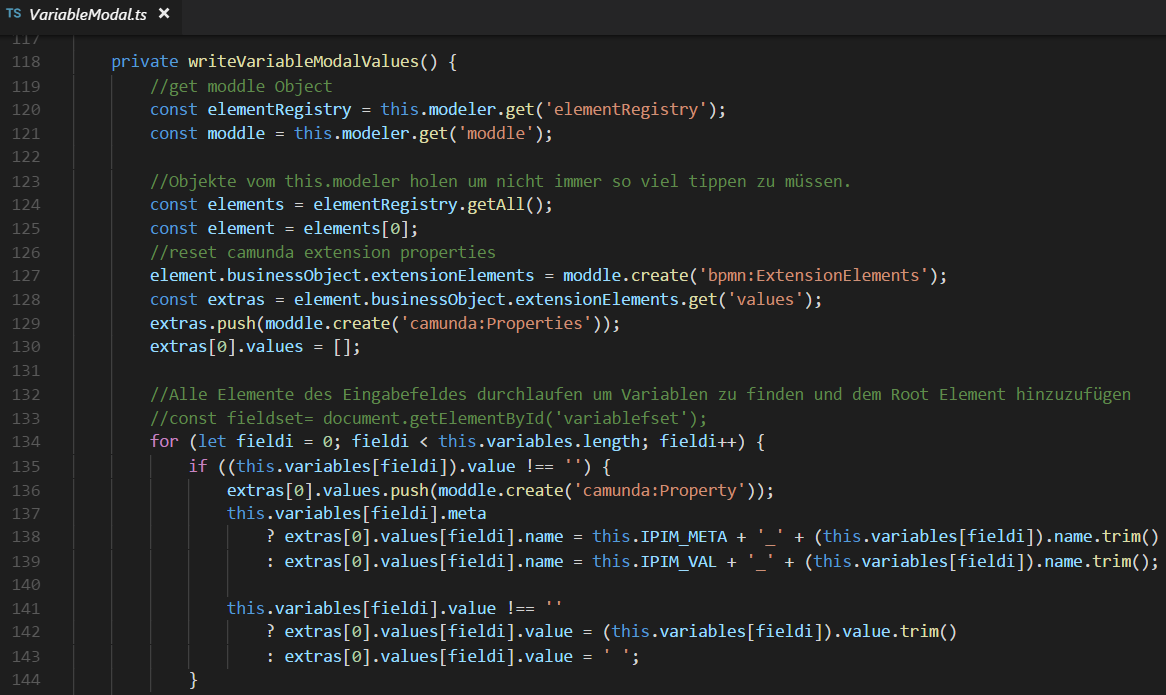


Abbildung 33: Setzen von Variablen

* Set Term

Die Logik zum Setzen von Termen ist im TermModal (siehe auch Kapitel 6.1.2.6) hinterlegt:

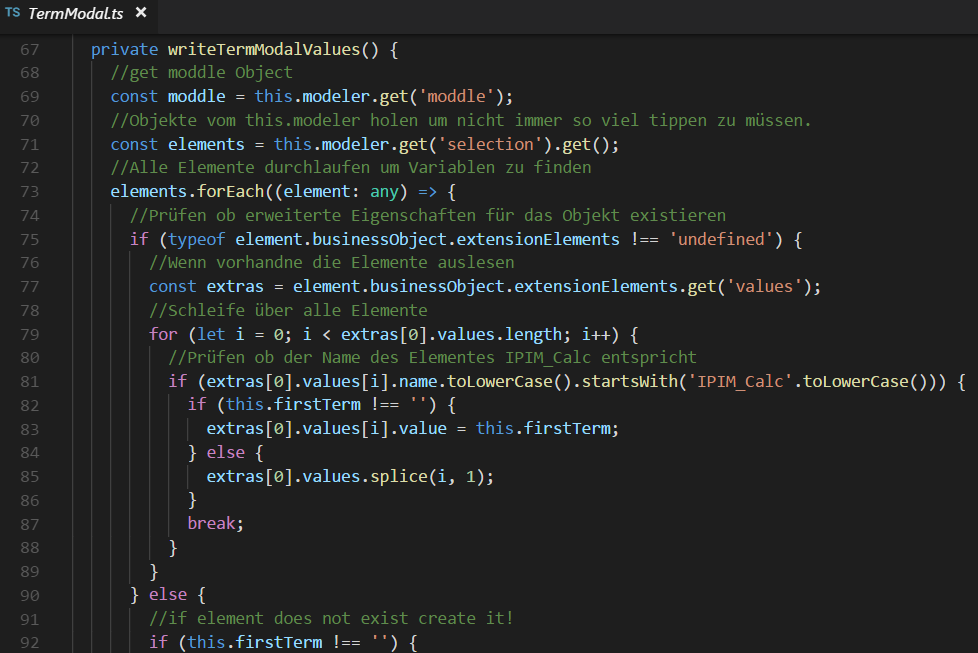


Abbildung 34: Setzen von Termen

* Evaluate Process

Prozesse können anhand der zugewiesenen Terme und Variablen für einzelne Prozessschritte ausgewertet werden. Die Logik hierzu befindet sich in modeler.component.ts und dem InputModal (siehe Kapitel 6.1.2.6):

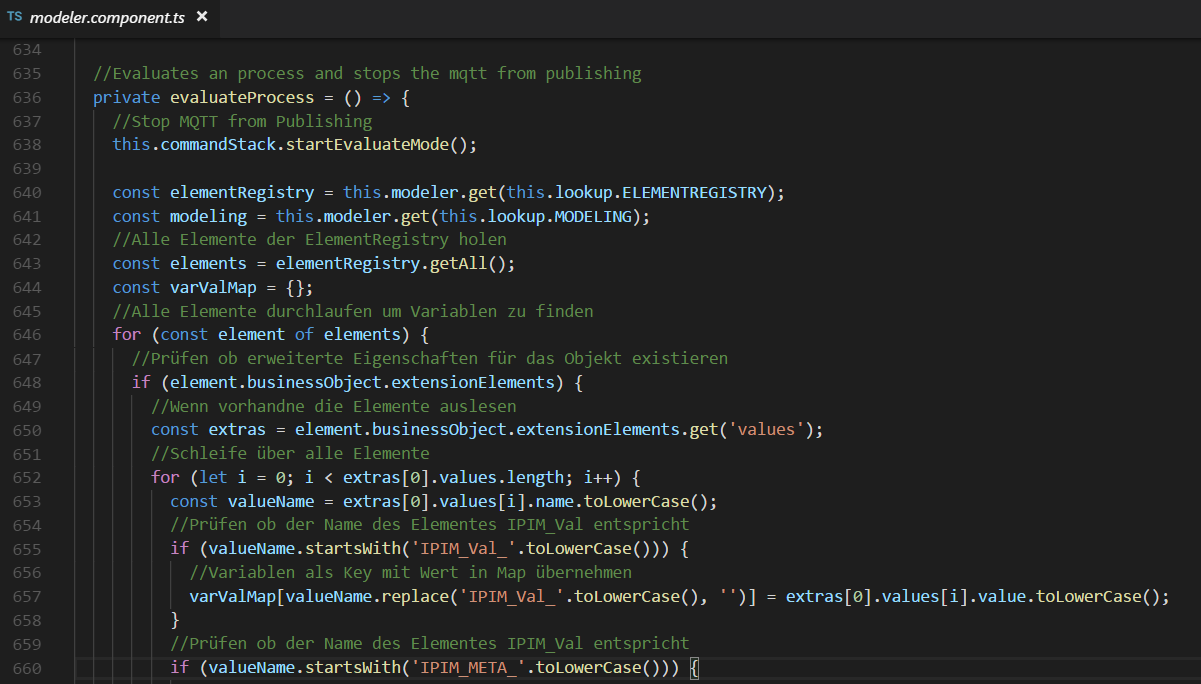


Abbildung 35: Evaluieren von einzelnen Modellen

* Start cascading Evaluation

Im Gegensatz zum Auswerten von einzelnen Modellpfaden besteht die Möglichkeit kaskadierend den aktuellen Prozess inklusive aller Suprozesse auszuwerten.

Zugeordnet hierzu ist das evalModal (siehe Kapitel 6.1.2.6). Das Öffnen dieses Modals und der zip-Download der beteiligten Modelle ist in modeler.component.ts implementiert:

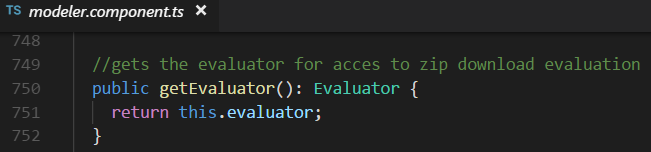


Abbildung 36: zip-Download der betroffenen Modelle beim kaskadierenden Evaluieren

Die Logik zum eigentlichen Evaluator befindet sich in einem eigenen Ordner, da diese etwas umfangreicher ist.

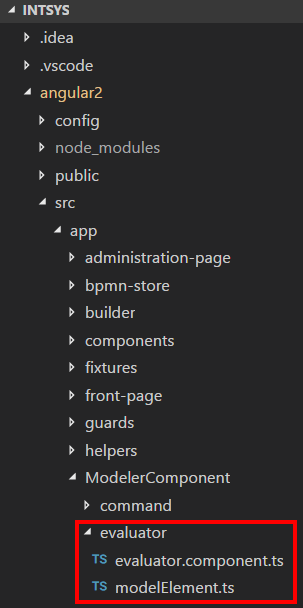


Abbildung 37: Dateistruktur zum kaskadierenden Auswerten

* Set Subprocess

Ein Subprozess kann über das SubProcessModal gesetzt werden (siehe auch Kapitel 6.1.2.6):

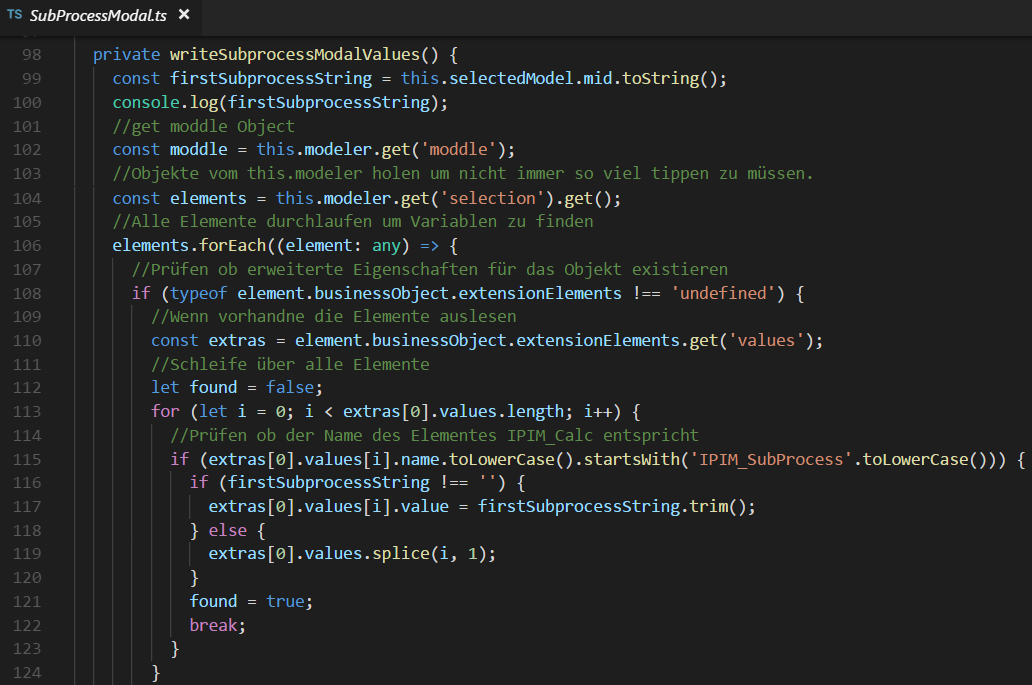


Abbildung 38: Setzen eines Subprozesses

* Open Model of Subprocess

Diese Funktion ist in modeler.component.ts implementiert:

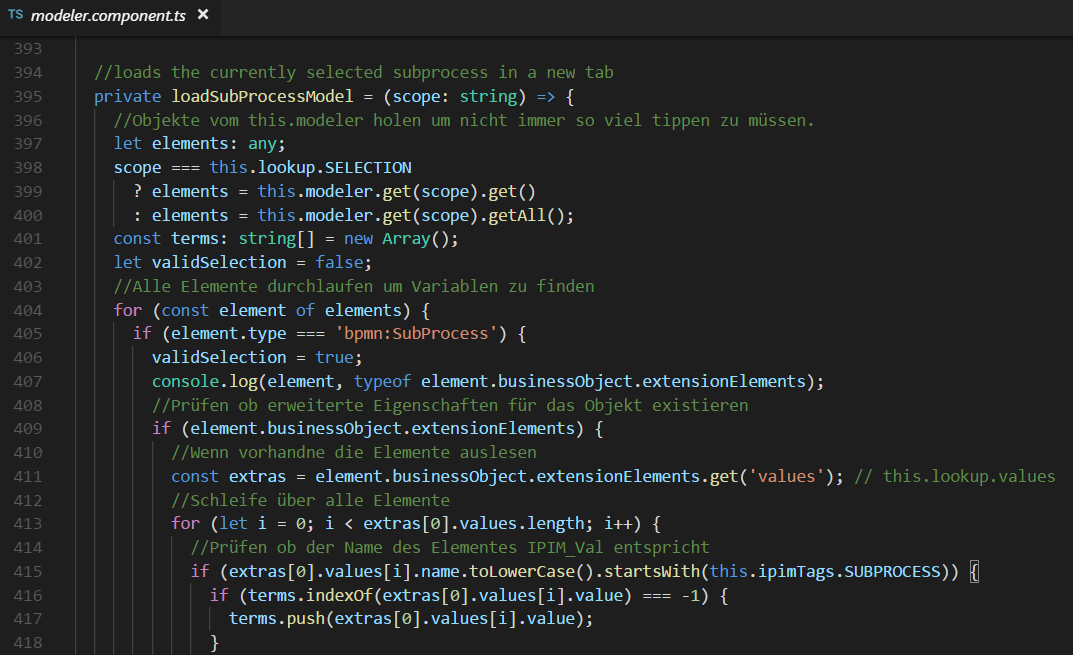


Abbildung 39: Öffnen eines Subprozesses in einem neuen Tab

* Highlight Elements

Diese Funktion ist in modeler.component.ts implementiert:

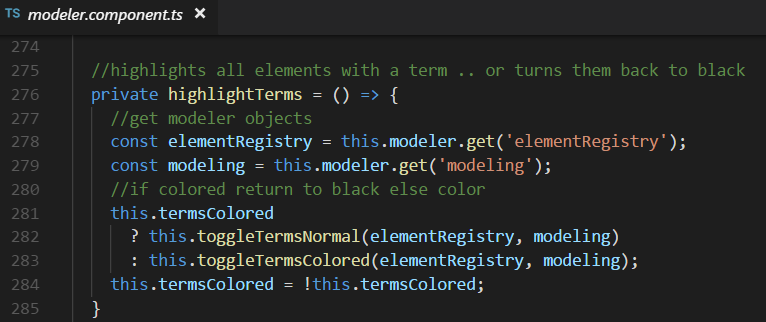


Abbildung 40: Farbliches Hervorheben von Prozesspfaden anhand ihrer zugewiesenen Terme

* Reset Diagram

Diese Funktion ist in modeler.components.ts implementiert:

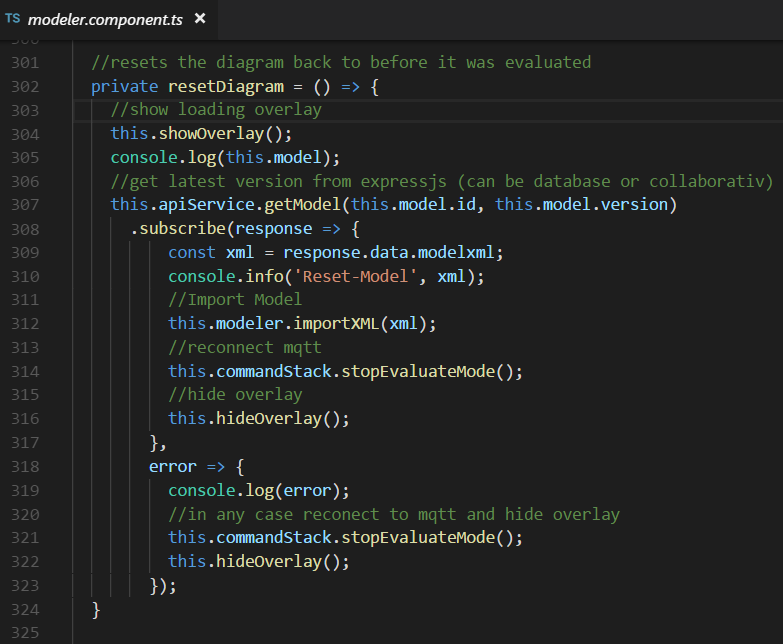


Abbildung 41: Reset des Diagramms

* See Processes references

Diese Funktion ist in modeler.components.ts implementiert:

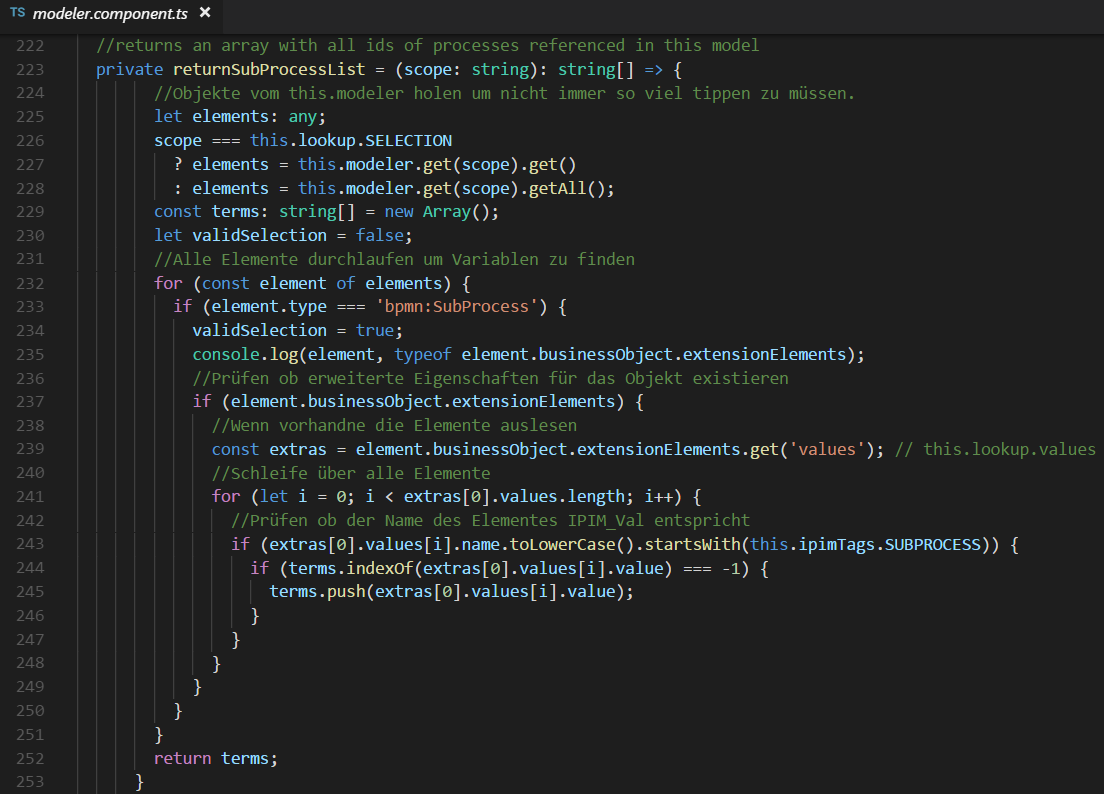


Abbildung 42: Anzeige aller Prozessreferenzen

### Administration Page

Mit der Admin-Berechtigung kann ein User über die Navigation Bar auf den Administrationsbereich zugreifen, andernfalls wird diese Option nicht angezeigt. Neben dem Absprung aus der Navigation Bar ist die Seite auch über die Sub-URL „administration-page“ aufrufbar. Dort angelangt kann dieser weitere Nutzer, Modelle, Rollen und Berechtigungen verwalten.

Die allgemeine Logik hierzu befindet sich im Ordner Administration Page.

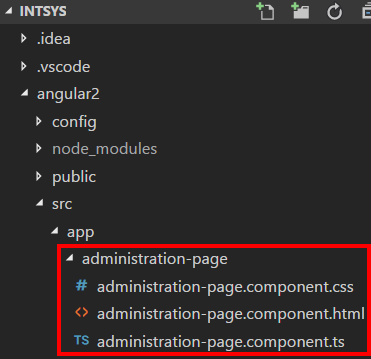


Abbildung 43: Coding zur Administration Page

#### Komponenten der Administration Page

Für jeden einzelnen Tab in der Administration Pagewurde mindestens eine Komponente realisiert.

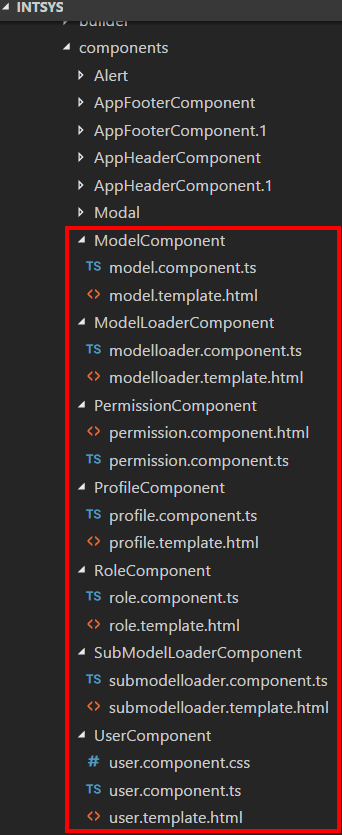


Abbildung 44: Struktur der Komponenten für die Administration Page

### Ändern des Farbschemas der Login Page

Bei Bedarf kann das Farbschema von KIP (grün) und IPIM (rot) gewechselt werden. Da dies für den User keine Standard-Funktionalität, sondern nur für Präsentationszwecke gedacht ist, kann dieser Wechsel nur im Coding durchgeführt werden. Welche Stellen hierfür relevant sind wird nachfolgend aufgezeigt.

#### Login

Sollte das Farbschema des Logins von KIP auf IPIM geändert werden müssen, so sind die Änderungen in der front-page.component.css an folgenden Stellen durchzuführen:

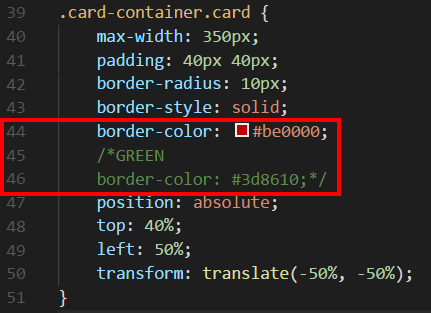


Abbildung 45: Änderung des Farbschemas der Login-Seite-Umrandung

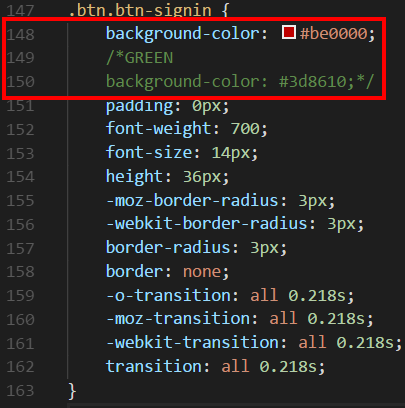


Abbildung 46:Änderung des Farbschemas der Login-Seite-Button

#### Schnittstelle GUI – Server

Die Verknüpfung zwischen der GUI und den Server-Abfragen befindet sich auf Angular-Seite in den services, allen voran in den api.service.ts. Dort ist für jede Abfrage dokumentiert, von welcher Funktion in der Oberfläche diese aufgerufen werden.

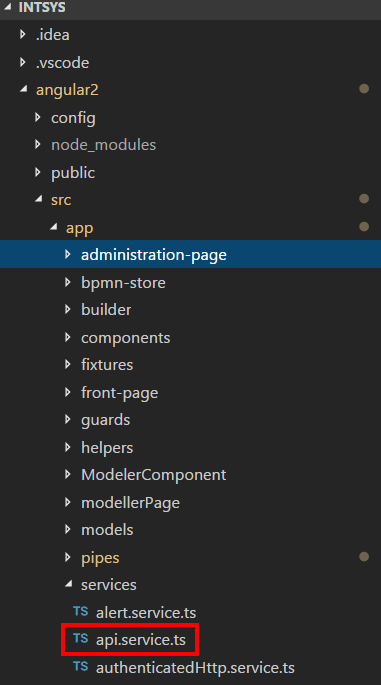


Abbildung 47: api.service.ts verknüpft die Oberfläche mit den Datenbank-Requests

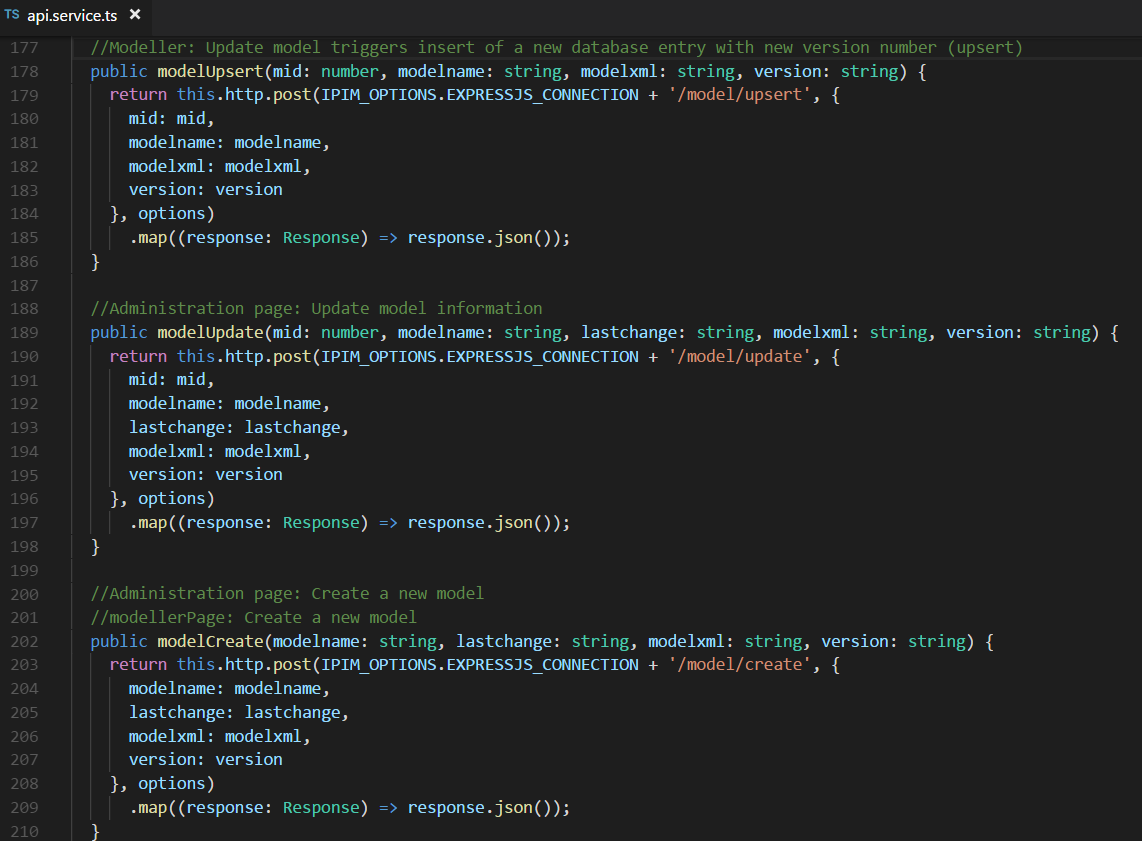


Abbildung 48: Dokumentation der Aufrufe in api.service.ts

#### Dependencies für die GUI

Alle eingebundenen Dependencies befinden sich im package.json-File im API-Ordner. Hier sind alle verwendeten Komponenten mit ihrer Version hinterlegt. Achtung: Die Datei package-lock.json wird beim Kompilieren automatisch erzeugt und beschreibt Aufbau der package.json und muss daher nicht angepasst werden!

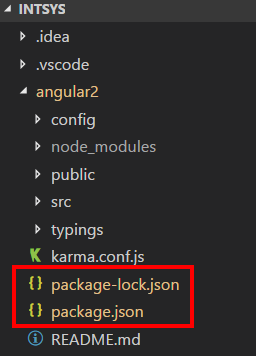


Abbildung 49: Dateistruktur für die Dependencies

## Aufbau des Express.js-Servers

Die Dateien des express.js-Servers liegen im API Ordner.

Die zentrale Datei ist express.js. Hier ist definiert, unter welcher URL der Server zu erreichen ist, welche anderen Dateien gemapped sind, wie sich ein User authentifiziert und wann ein User ausgeloggt wird.

## User.js, permission.js, model.js, partmodel.js, profile.js und role.js

Alle o.g. .js-Dateien beinhalten jede für sich im Wesentlichen die CRUD Endpoints.

In jeder der Dateien gibt es einen „all“ Request, der alle angeforderten Daten als Array zurückgibt.

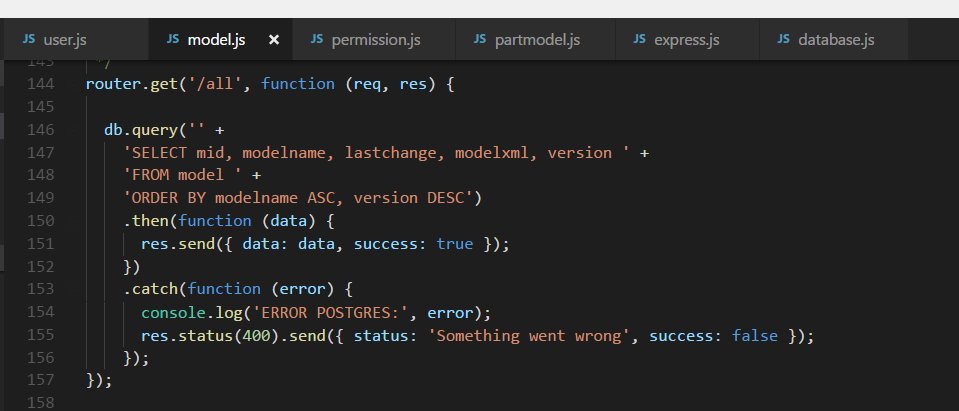


Abbildung 50: getAll-Request für Model in model.js

„.get“ ist dabei die HTTP-Methode, für die die Middlewarefunktion angewendet wird. „/all“ in den Klammern spiegelt den Pfad wieder, für den die Middlewarefunktion angewendet wird. „req“ ist das HTTP-Anforderungsargument zur Middlewarefunktion, die nach der geltenden Konvention als "req" (request) bezeichnet wird. „res“ ist das HTTP-Antwortargument zur Middlewarefunktion, die nach der geltenden Konvention als "res" (response) bezeichnet wird. Mit dem Select Statement werden alle Modelle aus der Model-Tabelle geladen und hinsichtlich des Modelnamens aufsteigend und der Versionsnummer absteigend als Array ausgegeben. Bei einem Fehler tritt die Fehlermeldung „Something went wrong“ ein.

Die Requests für Create, Update und Delete sind nach dem gleichen Prinzip aufgebaut. Sie unterscheiden sich im Wesentlichen jedoch dahingehend, dass sie kein „get“-Request, sondern ein „post“-Request beinhalten, da Daten in der Datenbank verändert und nicht nur ausgelesen werden. Bei Create wird ein neuer Datensatz angelegt, Update ändert den Datensatz und Delete löscht ihn.

Folgende Abbildung zeigt, welche die entsprechenden .js-Dateien im API Ordner sind.

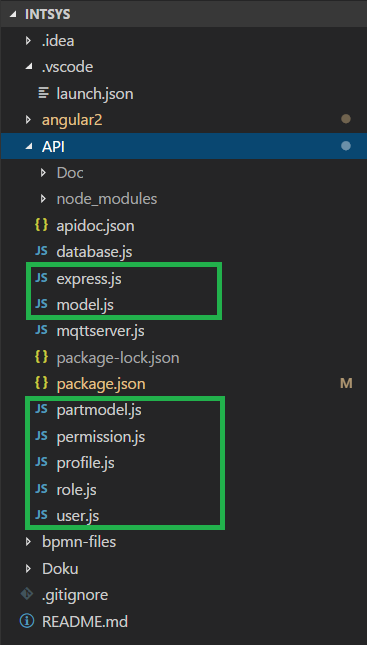


Abbildung 51: Aufbau des Express.js-Servers

## Aufbau des MQTT-Servers

Der Aufbau und die Logik des MQTT-Servers ist im API Ordner in der „mqttserver.js“ beschrieben.

Die Hauptaufgabe des MQTT-Servers ist hierbei das Verbinden der verschiedenen User über eine Gemeinsame Schnittstelle, welche es ermöglicht Modelle auszutauschen. Hierzu wird das Standard MQTT Protokoll verwendet. Dies bedeutet, dass sich jeder Interessent auf ein Topic Subscriben kann. Ist dies passiert kann er Nachrichten zu diesem Topic senden und empfangen.

Diese Funktionalität wird innerhalb des Projekts auf 2 Arten genutzt. Einerseits dient es dazu die Änderungen, welche an derzeit geöffneten Modellen vorgenommen werden, auszutauschen. Zum anderen werden auch Steuerdaten im Hintergrund übertragen, die es der expressJS Schnittstelle ermöglichen die Anzahl der gleichzeitig arbeitenden User zu erfassen und das Speichern von Modellen zu koordinieren.

Für den Rahmen dieses Projekts wurde das bestehende Standardprotokoll nur geringfügig modifiziert, sodass es möglich ist bereits beim Subscriben das ggf. bereits bearbeitete aber noch nicht gespeicherte Modell zu erhalten.

# Tests

Aus Gründen der Qualitätssicherung ist es für jede Anwendung ratsam, automatische Tests zu implementieren. Dies erfolgte als Teil des Projekts überwiegend im Fokus des API-Services und ist im angular2-Ordner unter test zu finden.

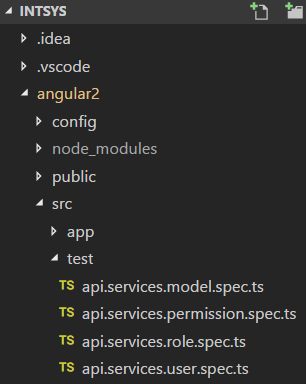


Abbildung 52: Dateistruktur zu den Tests

In den Tests wurden jeweils mindestens die Basic Anfragen wie: create, update und delete abgefragt. Damit die im realen Szenario entstehenden Server-Abfragen imitiert werden können, wurde in allen Tests ein Fake-Backend eingebaut, welcher die Antworten aus dem Server ausgibt.

Die Abbildung 51 zeigt am Beispiel der user.js, wie solch ein Fake-Backend aufgesetzt wird und die Abbildung 52 zeigt wie solch ein für Fake-Response letzendlich verwendet wird.

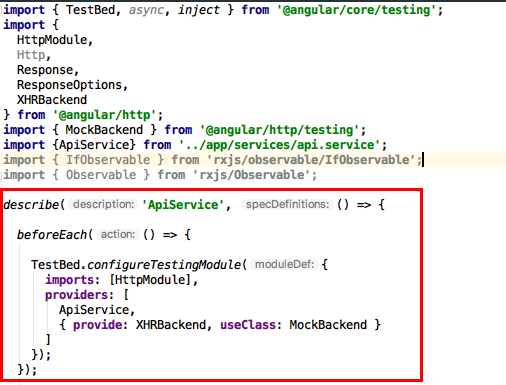


Abbildung 53: Aufsetzen von einem Fake-Backend



Abbildung 54: Verwendung des Fake-Backends