**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: FIXME.08.2018

Betreuer:

Daniel Hilpoltsteiner

Prof. Dr. rer. oec. Christian Seel

# Inhaltsverzeichnis

[Inhaltsverzeichnis II](#_Toc522553047)

[Abbildungsverzeichnis IV](#_Toc522553048)

[1 Allgemeine Beschreibung 1](#_Toc522553049)

[1.1 Rahmenbedingungen 1](#_Toc522553050)

[1.2 Entwicklerwerkzeuge 1](#_Toc522553051)

[2 Technischer Aufbau 3](#_Toc522553052)

[2.1 Datenbank 3](#_Toc522553053)

[2.2 MQTT 5](#_Toc522553054)

[2.3 Express.js 7](#_Toc522553055)

[2.4 Angular Frontend 7](#_Toc522553056)

[3 Installationsanleitung 8](#_Toc522553057)

[3.1 GitKraken 8](#_Toc522553058)

[3.2 Visual Studio Code 8](#_Toc522553059)

[3.3 express.js 9](#_Toc522553060)

[3.4 Postgres 9](#_Toc522553061)

[3.5 pgAdmin 10](#_Toc522553062)

[3.6 Open VPN 10](#_Toc522553063)

[4 Lokale und Server-Datenbank 11](#_Toc522553064)

[4.1 Anlage einer lokalen Datenbank 11](#_Toc522553065)

[4.2 Wechsel zwischen lokaler und Server-Datenbank 12](#_Toc522553066)

[5 Bedienungsanleitung für Entwickler 13](#_Toc522553067)

[6 Aufbau des Codings 16](#_Toc522553068)

[6.1 Zusammenhang von Funktionalitäten der Oberfläche mit dem Coding 16](#_Toc522553069)

[6.1.1 Login 16](#_Toc522553070)

[6.1.2 Modeller 17](#_Toc522553071)

[6.1.2.1 modellerPage 17](#_Toc522553072)

[6.1.2.2 Model 19](#_Toc522553073)

[6.1.2.3 Versionierung 20](#_Toc522553074)

[6.1.2.4 Navigation Bar 22](#_Toc522553075)

[6.1.2.5 Modeller-Palette 23](#_Toc522553076)

[6.1.2.6 Modale 25](#_Toc522553077)

[6.1.3 Administration Page 28](#_Toc522553078)

[6.1.3.1 Navigation Bar 30](#_Toc522553079)

[6.1.3.2 Komponenten der Administration Page 30](#_Toc522553080)

[6.1.4 Ändern des Farbschemas der Login Page 31](#_Toc522553081)

[6.1.4.1 Login 32](#_Toc522553082)

[6.1.4.2 TODO: weitere Stellen notwendig? 33](#_Toc522553083)

[6.2 Schnittstelle und Datenbank 34](#_Toc522553084)

[7 Benutzerhandbuch 36](#_Toc522553085)

[7.1 Allgemeiner Aufbau 36](#_Toc522553086)

[7.2 Modeller: Modell anlegen, bearbeiten (löschen?) 36](#_Toc522553087)

[7.3 Administration Page: Stammdaten anlegen, bearbeiten, löschen 36](#_Toc522553088)

[8 Arbeitspakete 37](#_Toc522553089)

[Literaturverzeichnis 38](#_Toc522553090)

# Abbildungsverzeichnis

[Abbildung 9: Starten des Node-Paketmanagers 14](#_Toc522553249)

[Abbildung FIXME: Login mit den Testdaten 15](#_Toc522553250)

[Abbildung 16: Definition von Model und User 20](#_Toc522553251)

[Abbildung 17: Versionierung von Modellen anhand eines scrum Beispiels 21](#_Toc522553252)

[Abbildung 18: File zur Versionierung 22](#_Toc522553253)

[Abbildung 19: Versionsnummer des Modells in der Navigation Bar 22](#_Toc522553254)

[Abbildung 20: Palette mit Icons für weitere Funktionen 24](#_Toc522553255)

[Abbildung 21: Dateien zur Palettenverwaltung 25](#_Toc522553256)

[Abbildung 22: Term Modal 26](file:///C:\Users\Jenzo\Documents\HAW%20Landshut\Studienprojekt\DeveloperGuide_ADAMO.docx#_Toc522553257)

[Abbildung 23: Variablen Modal 26](file:///C:\Users\Jenzo\Documents\HAW%20Landshut\Studienprojekt\DeveloperGuide_ADAMO.docx#_Toc522553258)

[Abbildung 24: Dateistruktur zu den Modalen 27](#_Toc522553259)

[Abbildung 25: Aufruf der Komponente aus dem Variablen Modal heraus 28](#_Toc522553260)

[Abbildung 26: Komponente zum Variablen Modal enthält den zugehörigen selector 28](#_Toc522553261)

[Abbildung 27: Administration page 29](#_Toc522553262)

[Abbildung 28: Coding zur Administration Page 30](#_Toc522553263)

[Abbildung 29: Struktur der Komponenten, welche für die Administration Page relevant sind 31](#_Toc522553264)

[Abbildung 30: Änderung des Farbschemas der Login-Seite – Umrandung 32](#_Toc522553265)

[Abbildung 31:Änderung des Farbschemas der Login-Seite - Button 33](#_Toc522553266)

[Abbildung 32: api.service.ts verknüpft die Oberfläche mit den Datenbank-Requests 34](#_Toc522553267)

[Abbildung 33: Dokumentation der Aufrufe in api.service.ts 35](#_Toc522553268)

# Tabellenverzeichnis

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

# Abkürzungsverzeichnis

ADAMO Adaptiver Modeller

BPMN Business Process Model and Notation

CMMN Case Management Model and Notation

DMN Decision Model and Notation

EFRE FIXME

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 Modellierungssprachen BPMN 2.0, CMMN 1.0 und DMN 1.1 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 (EFRE 🡪 KIP oder? FIXME) 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 ein Publisher/Subscriber Modell 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. Dazu wurde für VS Code das Git-Plugin installiert, mit dessen Hilfe die Git- Funktionen innerhalb der Entwicklungsumgebung genutzt werden konnten.

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

# Technischer Aufbau

ToDo Markus: Irgendwas schlaues darüber erzählen, wie Camunda generell arbeitet

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

Abbildung FIXME Three Tier Architecture

Wie der Name bereits sagt, besteht diese Architektur aus drei Schichten. Die unterste Schicht ist die Datenbankschicht, die mit Hilfe von PostgreSQL (kurz Postgres) realisiert wird. Diese Schicht sorgt für die Persistenz der Daten. Die mittlere Schicht ist die Applikationsschicht. Diese wird mit Hilfe von zwei Servern realisiert. 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 MQTT eingesetzt. Dieser ermöglicht die Funktionen des Sucscriben auf ein/mehrere Modelle sowie das kollaborative Arbeiten an Modellen. Die oberste Schicht ist die Präsentationsschicht. Diese wird durch zwei getrennte Clients mit graphischer Benutzungsschnittstelle realisiert.

Der erste Client dient der Pflege des gesamten Datenbestands auf der Administrationsseite. Diesen Client sieht ein Anwender nur mit den entsprechenden Rechten (Userprofil Administrator). Der Anwender kann hier die Stammdaten Modell, User, Rolle, Berechtigung, Teilmodell und Userprofil sowohl anlegen, editieren als auch löschen.

Der zweite Client 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 FIXME zu finden.

Der Aufbau der Datenbank folgt dem folgenden 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 den Fremdschlüssel der ID 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 FIXME veranschaulicht das voran erklärte Datenbankschema mit seinen Beziehungen:

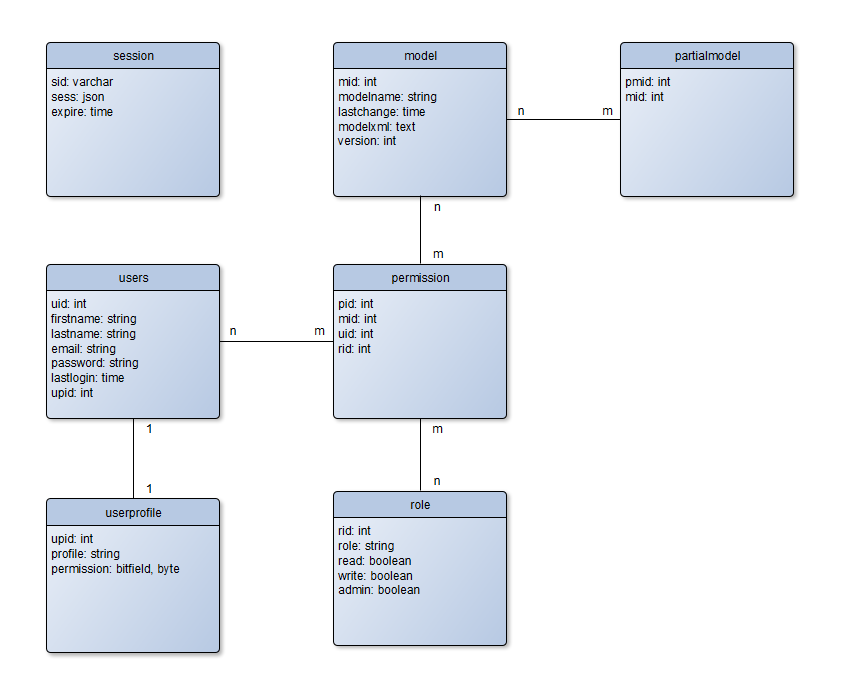


Abbildung FIXME: Datenbanktabellen und ihre Beziehungen zueinander

## MQTT

Für das Studienprojekt wurde ein MQTT Server für das Austauschen von Nachrichtenprotokollen aufgesetzt, welcher es den Usern unter anderem ermöglicht für bestimmte Modelle zu „subscriben“. Außerdem können damit 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.

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

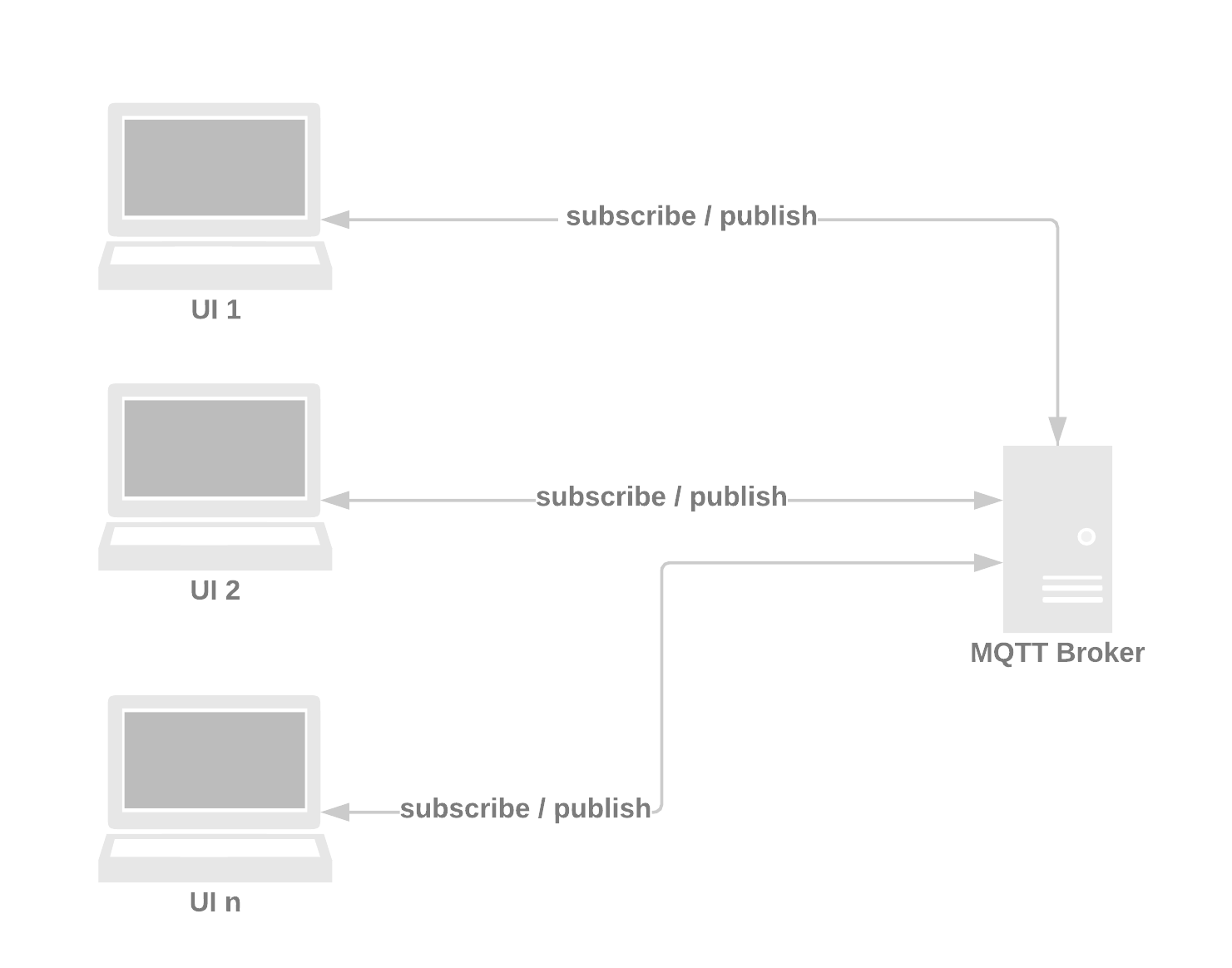


Abbildung FIXME: Publish/Subscriber Modell bei MQTT

Die MQTT Server Datei (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.

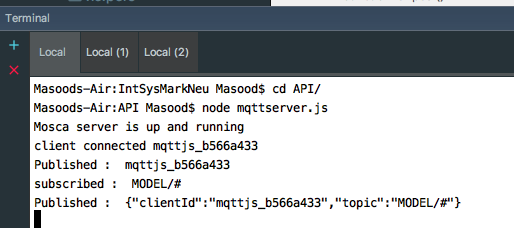


Abbildung FIXME: Starten des MQTT brokers über die Konsole

## 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.

ToDo

## Angular Frontend

ToDo

# Installationsanleitung

## 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/In tSys.git.

Dafür GitKraken nach erfolgreichem Download und Installation starten und genannte URL unter

„File“ -> „Clone Repo“

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

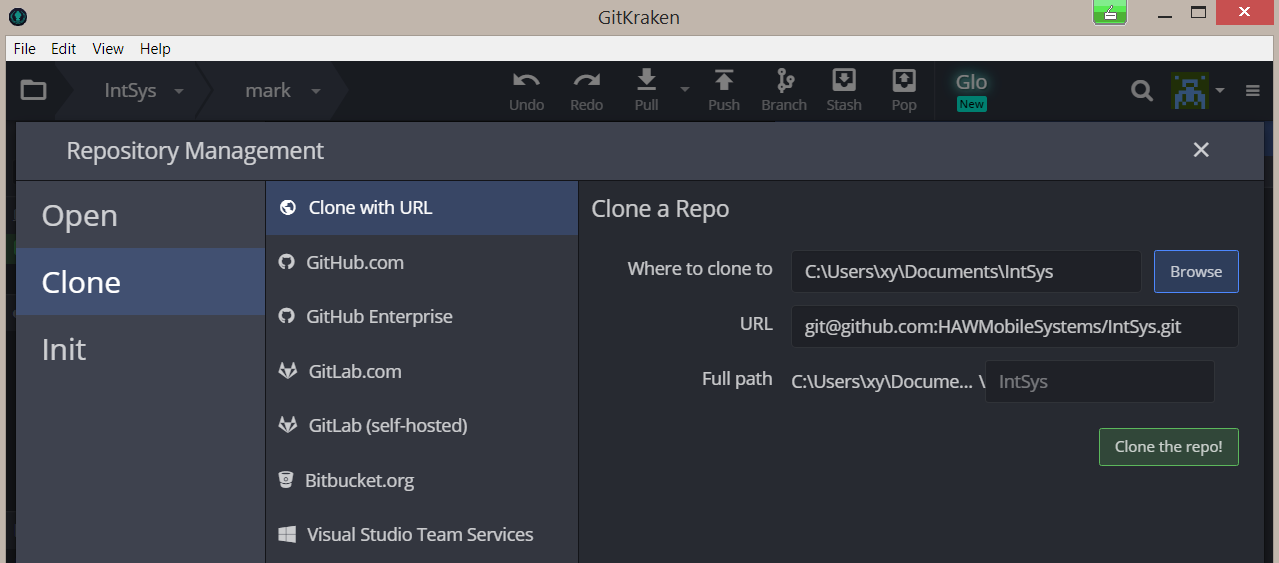


Abbildung FIXME: Klonen eines Repositories mittels GitKraken

## 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 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.

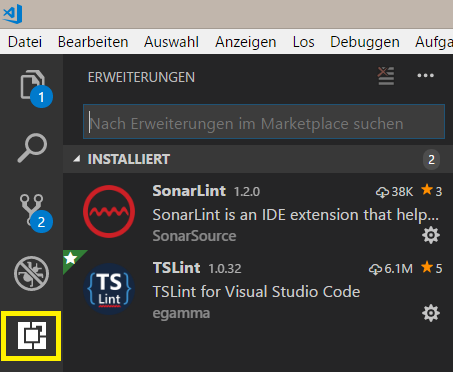


Abbildung FIXME: Installation von zusätzlichen Komponenten

## express.js

Für die Realisierung des Servers und weiterer auf JavaScript basierender Webentwicklungen durchzuführen ist node.js ein hilfreiches Framework, weshalb beim Modeller die Erweiterung express.js im Einsatz. Der Download steht unter http://expressjs.com/de/ zur Verfügung, wobei die bereitgestellte Installationsanleitung verwendet werden kann: http://expressjs.com/de/starter/installing.html. Die Installation kann im Terminal von Visual Studio Code durchgeführt werden.

TODO: Schritte im Terminal erklären

## Postgres

Optional, wird nur benötigt falls eine lokale Datenbank genutzt werden soll.

Ein Download von Postgres ist unter <https://www.postgresql.org/download/> möglich.

## pgAdmin

TODO

## 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

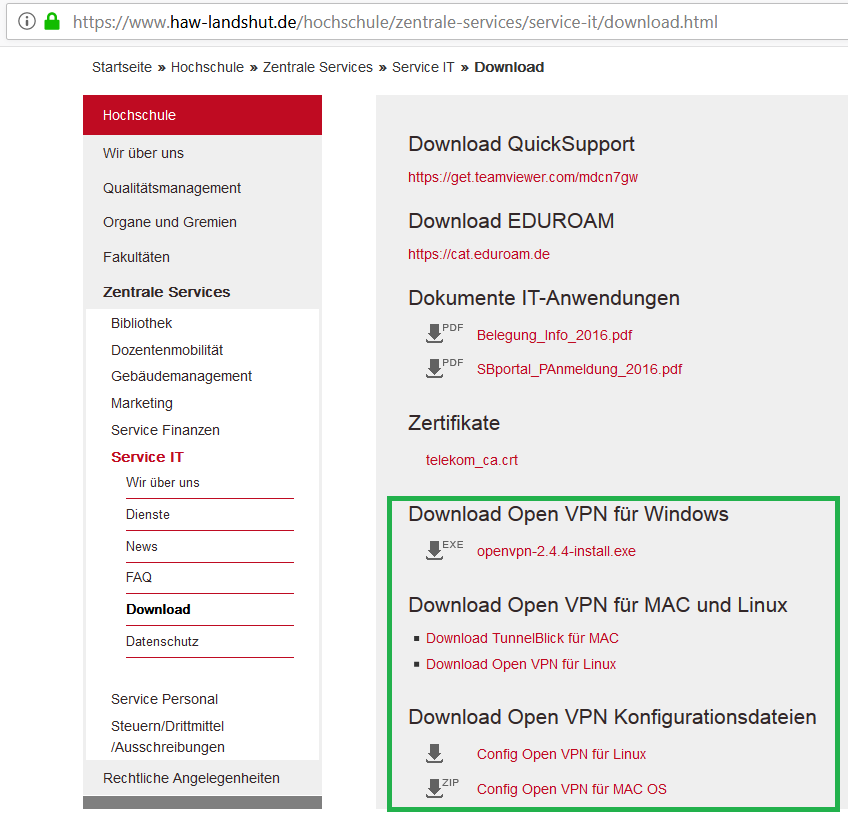


Abbildung FIXME: Download-Möglichkeiten von Open VPN der HAW Landshut

# 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.

CREATE DATABASE ipim

WITH

OWNER = postgres

ENCODING = 'UTF8'

LC\_COLLATE = 'German\_Germany.1252'

LC\_CTYPE = 'German\_Germany.1252'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1;

FIXME

## 4.2 Wechsel zwischen lokaler und Server-Datenbank

TODO

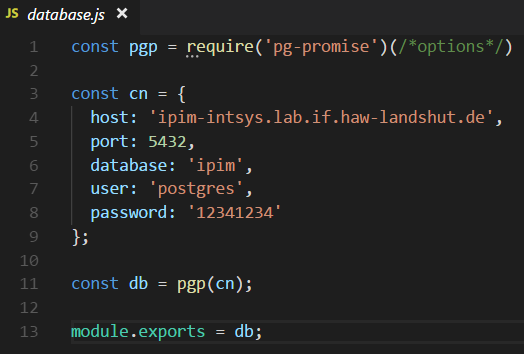


Abbildung FIXME: Verbindungsdaten im File database.js

# Bedienungsanleitung für Entwickler

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

1. Entweder die lokale Datenbank aktiv haben (siehe 4.2) oder mit VPN zur Server-Datenbank verbinden
2. Visual Studio Code starten
3. Index.js 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 index.js ausführen, um startup, routing und andere Basisfunktionen zu starten. Falls eine Fehlermeldung bezüglich Timeout erscheint, sollte die VPN-Verbindung geprüft werden

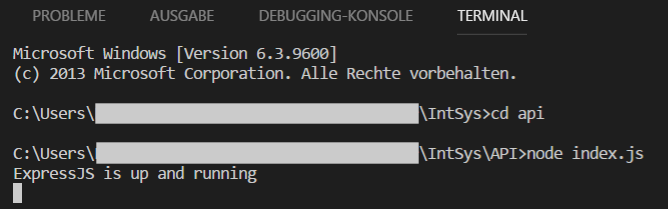


Abbildung FIXME: index.js starten

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.

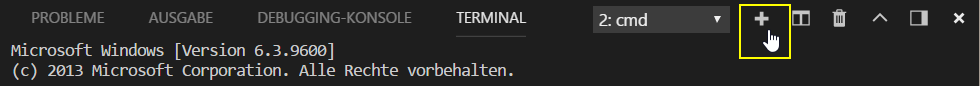


Abbildung FIXME: Öffnen einer neuen Konsole

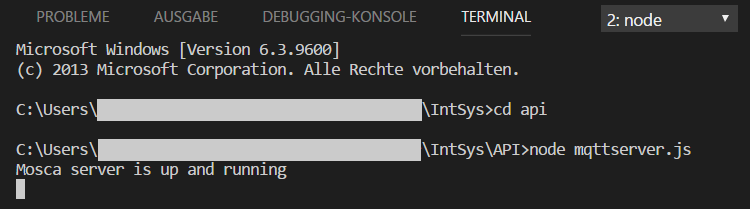


Abbildung FIXME: Starten des MQTT-Servers

1. MQTT 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 erscheinen Erfolgsmeldungen, 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

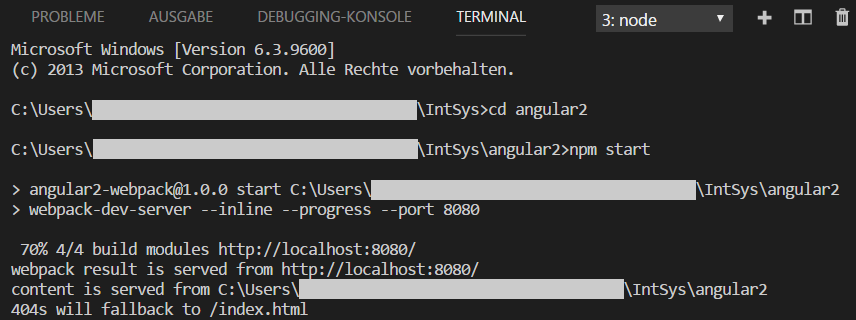


Abbildung 9: Starten des Node-Paketmanagers

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

User: 1

Passwort: 12341234

Abbildung FIXME: Login mit den Testdaten

# Aufbau des Codings

Prinzipiell ist die Dateistruktur des Codings in zwei für die Entwicklung relevante Ordner aufgeteilt: angular2 und API. Im angular2-Ordner befinden sich alle für die Oberfläche 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.

## Zusammenhang von Funktionalitäten der Oberfläche mit dem Coding

### Login

Die Login-Seite ist unter der Sub-URL „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.

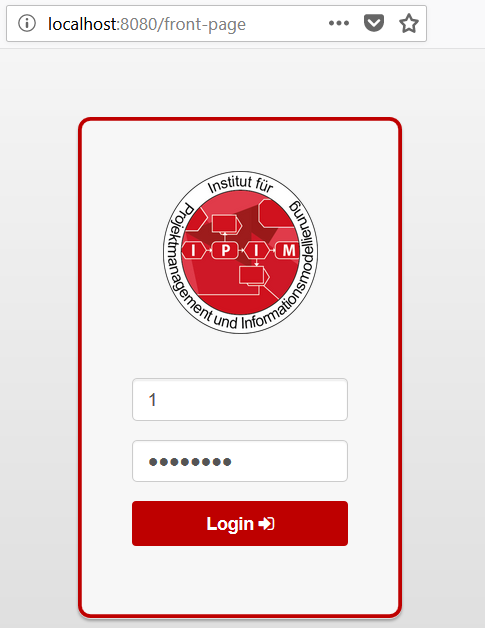
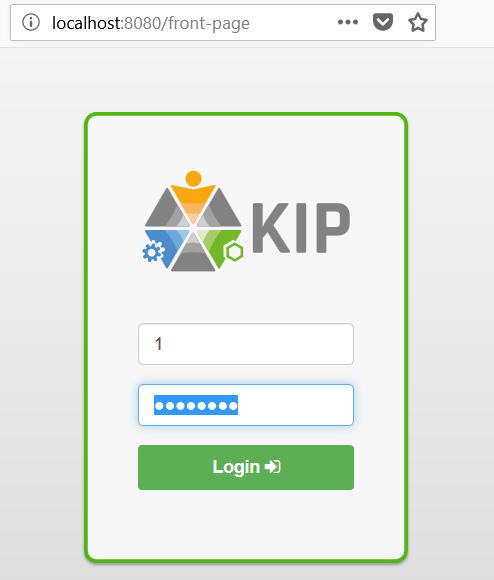


Abbildung FIXME: Login Page im KIP-Design vs. Login Page im IPIM-Design

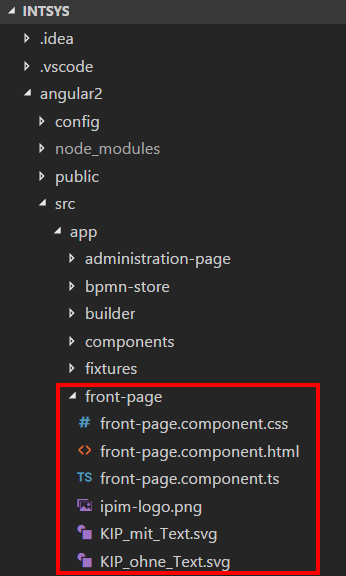


Abbildung FIXME: Front Page Struktur in VS Code

### Modeller

Nach dem Login wird der User auf den Modeller geleitet, welches sich in die modellerPage und den eigentlichen Modeller aufgliedert. Verfügbar ist dieser unter der Sub-URL „modeler“

#### modellerPage

Die vorgeschaltete modellerPage ist eine Übersichtsseite für alle vorhandenen Modelle, von denen eins in der gewünschten Version ausgewählt und geladen oder alternativ ein neues Modell erstellt werden kann.

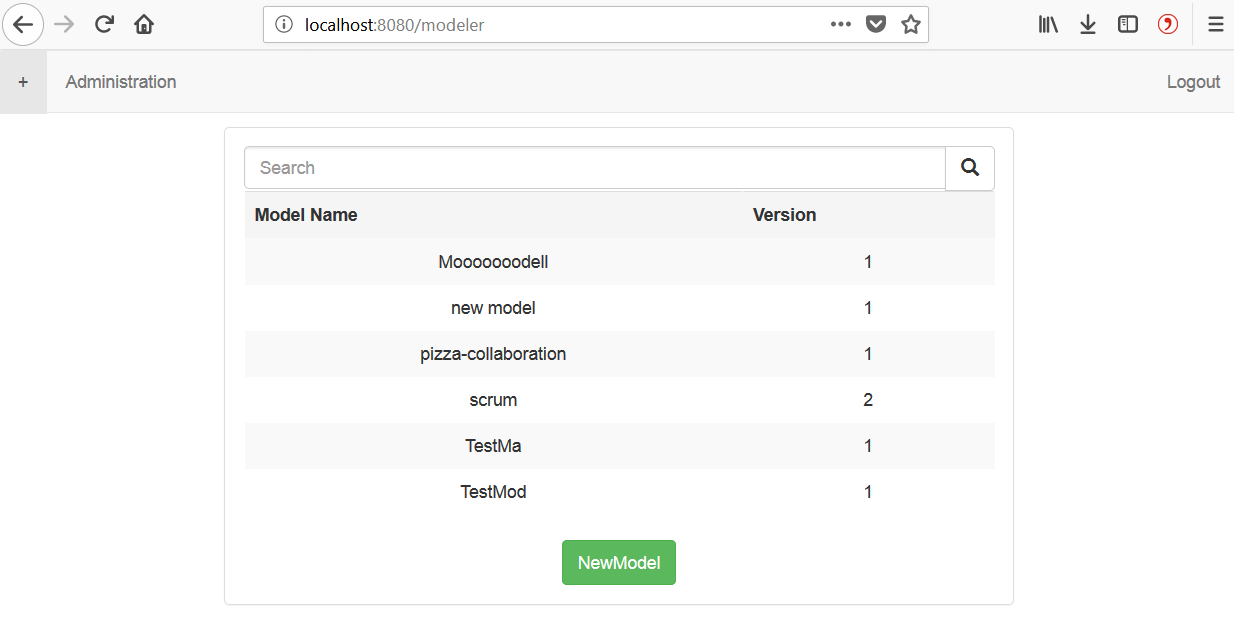


Abbildung FIXME:

Das zugehörige Coding findet sich im Paket modellerPage.

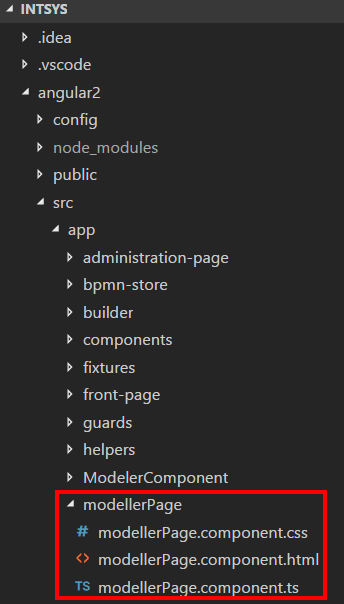


Abbildung FIXME: modellerPage Struktur in VS Code

Die Seite teilt sich auf in die Navigation Bar und eine Liste von Modellen. Ein Modell ist wiederum im File „model.ts“ klassifiziert.

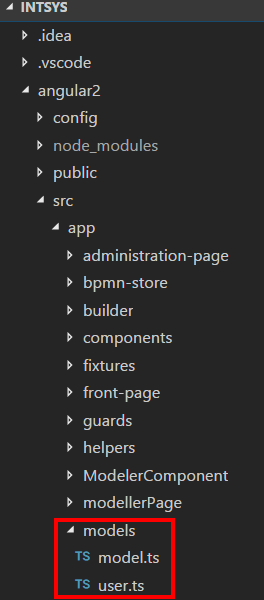


Abbildung FIXME: Definition von Model und User in VS Code

#### Model

Die Seite teilt sich auf in die Navigation Bar und eine Liste von Modellen. Ein Modell ist wiederum im File „model.ts“ klassifiziert.

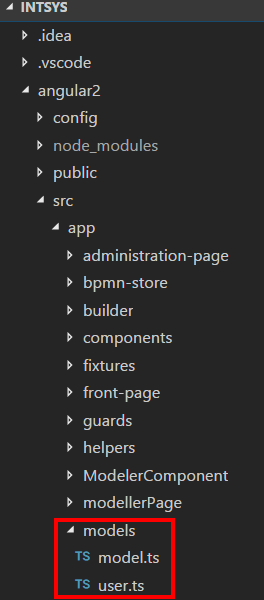


Abbildung 16: Definition von Model und User

#### Versionierung

Sobald ein Modell in der modellerPage ausgewählt wurde, werden 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 (beispielsweise 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

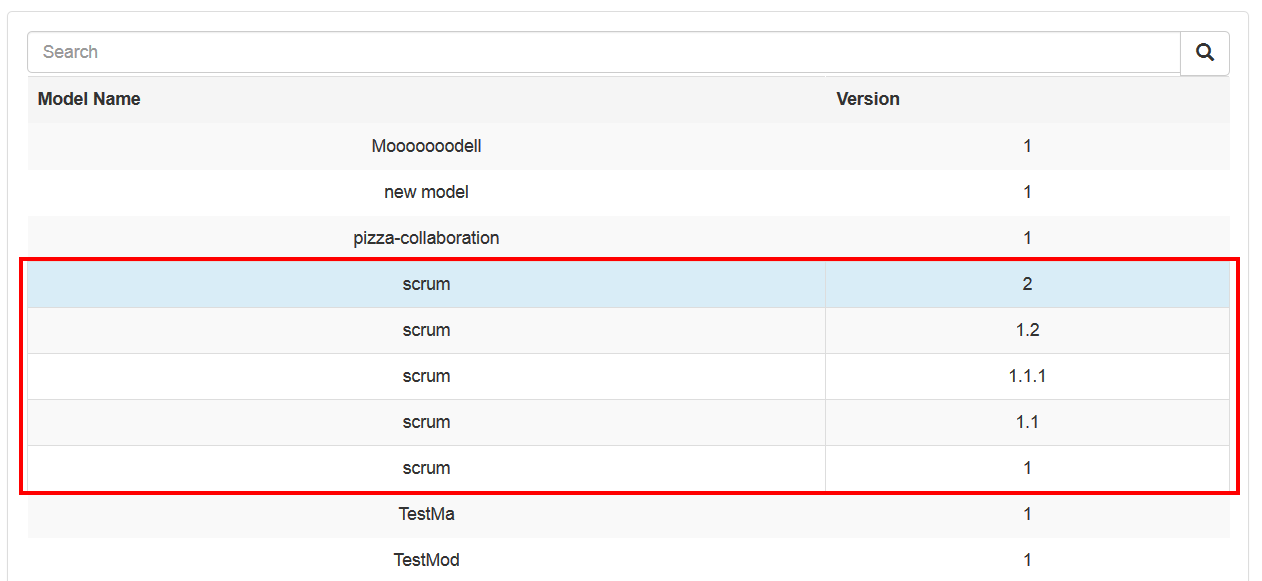


Abbildung 17: Versionierung von Modellen anhand eines scrum Beispiels

Die Logik zu der Berechnung der Versionsnummer findet sich in version.pipe.ts.

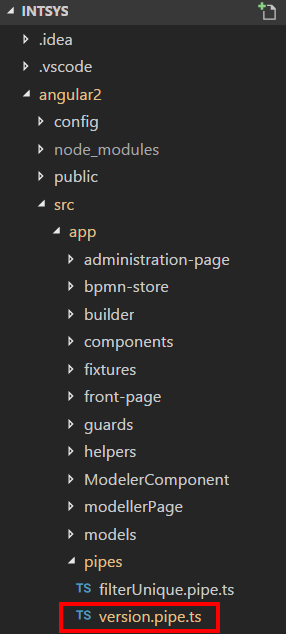


Abbildung 18: File zur Versionierung

#### Navigation Bar

In der Navigation Bar wird die Versionsnummer eines geöffneten Modells ebenfalls mit angezeigt.



Abbildung 19: Versionsnummer des 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.

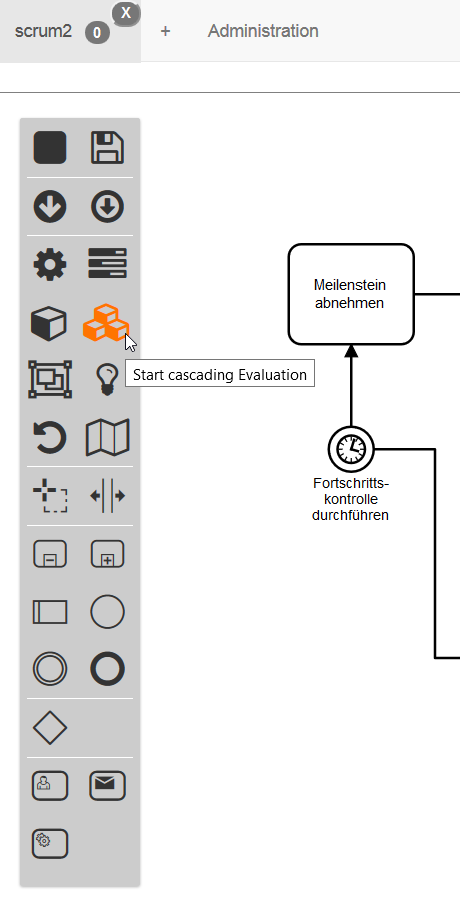


Abbildung 20: Palette mit Icons für weitere Funktionen

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 bereits von Camunda zur Verfügung gestellten und angepassten Icons hinterlegt, in zweiterer die für zusätzliche Funktionen benötigten Icons. Die Icons werden einem Kommando zugeordnet und eine Aktion bei der Auswahl hinterlegt, wobei das Kommando in der commandstore.service.ts definiert ist.

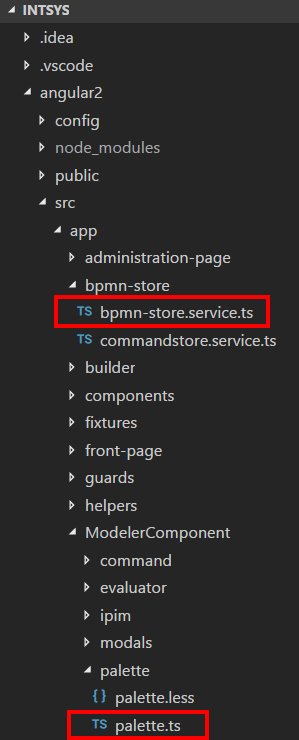


Abbildung 21: Dateien zur Palettenverwaltung

#### Modale

Zu vielen Icons in der Palette öffnet sich ein Modal, welches dem User weitere Optionen bietet. Im Detail sind dies das 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:

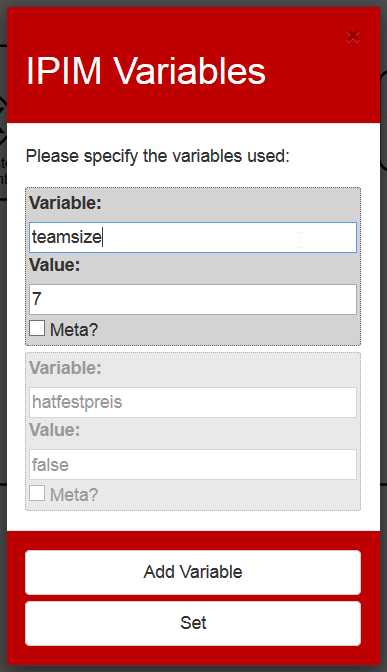
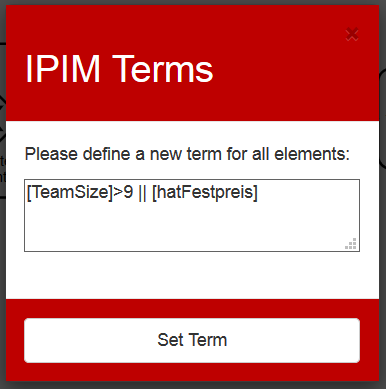


Abbildung 22: Term Modal

Abbildung 23: Variablen Modal

Die übergreifende Oberflächengestaltung aller Modale ist in der Modal.css, die Farbgestaltung in Modal.less (Ordner components) 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.

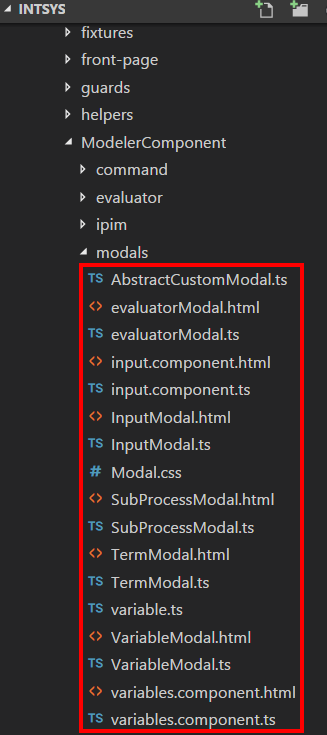


Abbildung 24: Dateistruktur zu den Modalen

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

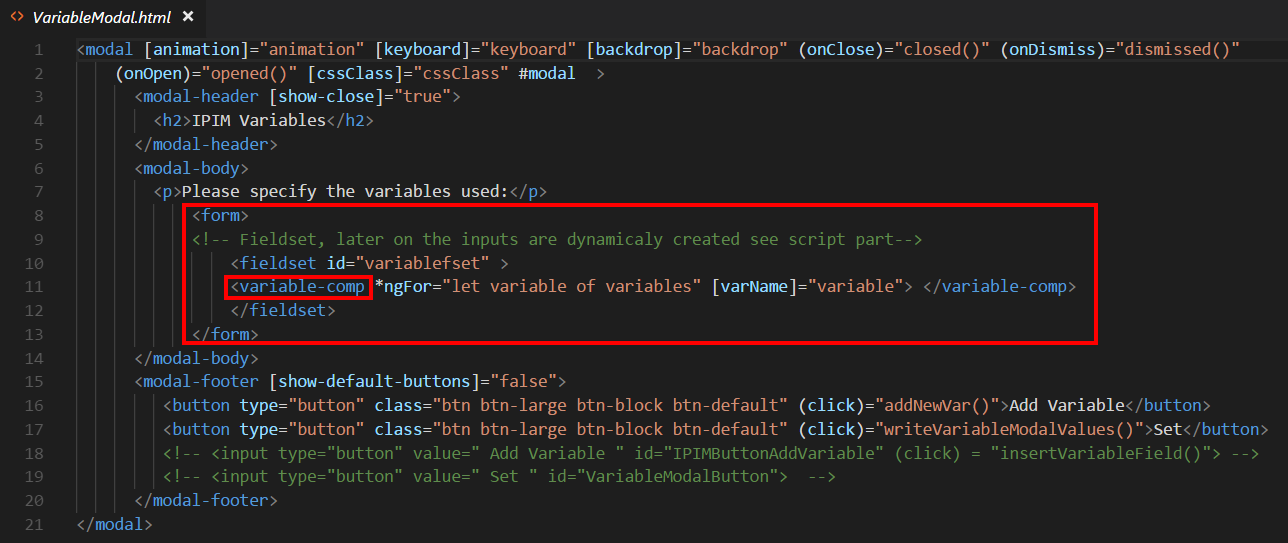


Abbildung 25: Aufruf der Komponente aus dem Variablen Modal heraus

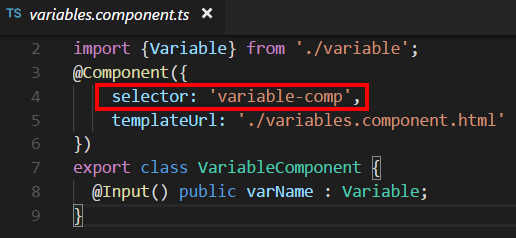


Abbildung 26: Komponente zum Variablen Modal enthält den zugehörigen selector

TODO: Einzelne Funktionen beschreiben? Evaluieren etc…Sobald fertig…

### Administration Page

Mit ausreichend 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.

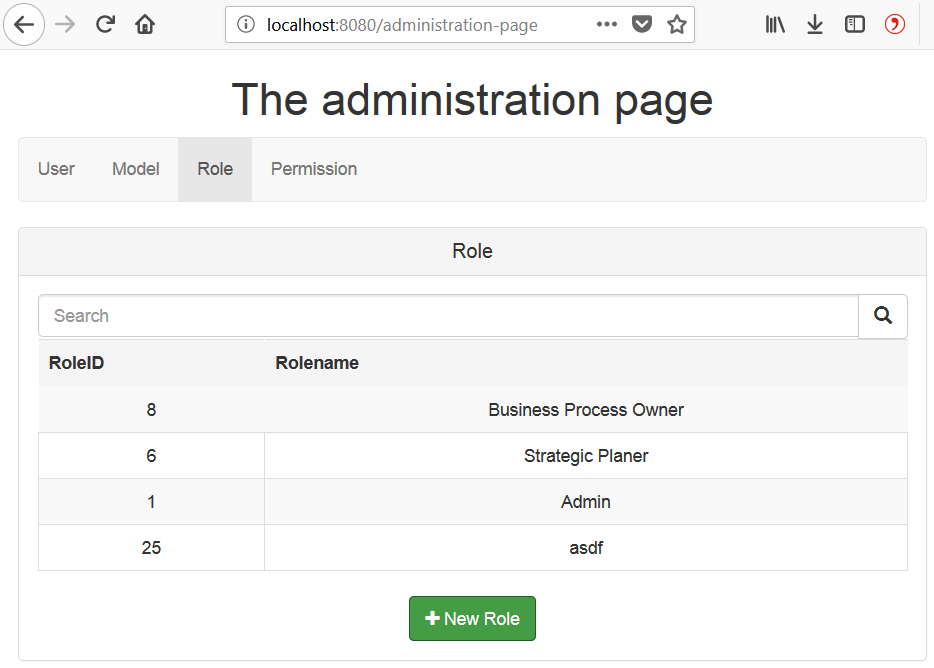


Abbildung 27: Administration page

Die allgemeine Logik hierzu befindet sich im Ordner 1.1.1.1 Administration Page.

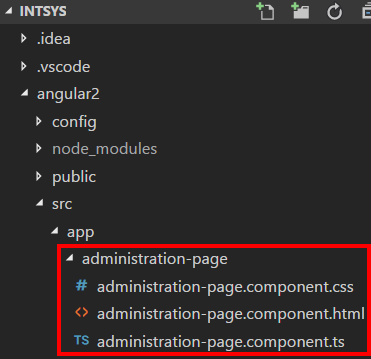


Abbildung 28: Coding zur Administration Page

#### Navigation Bar

Siehe Kapitel 6.1.2.4.

#### Komponenten der Administration Page

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

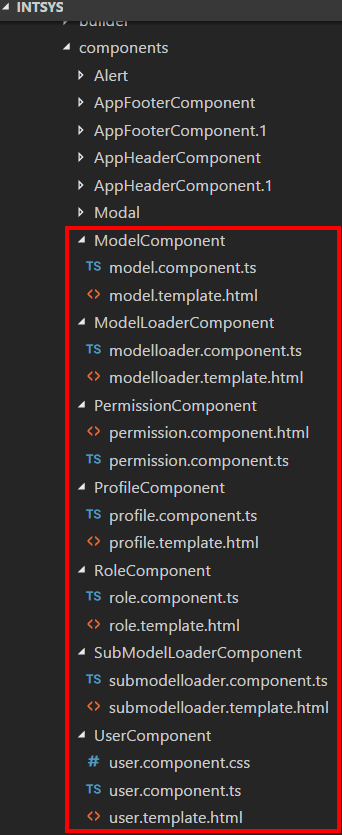


Abbildung 29: Struktur der Komponenten, welche für die Administration Page relevant sind

### 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:

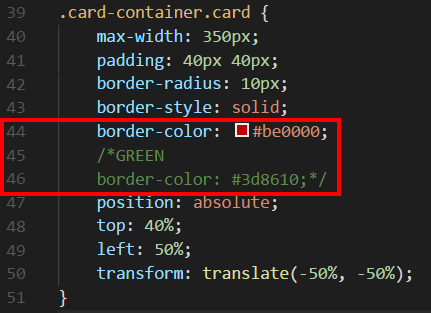


Abbildung 30: Änderung des Farbschemas der Login-Seite – Umrandung

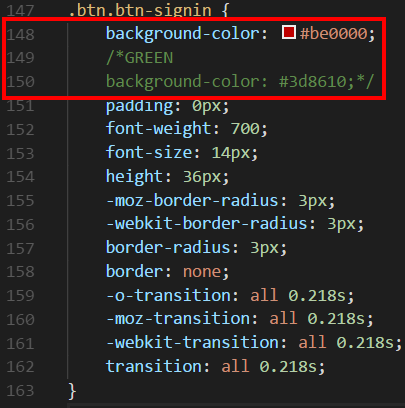


Abbildung 31:Änderung des Farbschemas der Login-Seite - Button

#### TODO: weitere Stellen notwendig?

## Schnittstelle und Datenbank

Die Verknüpfung zwischen Oberfläche und Datenbankabfragen 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.

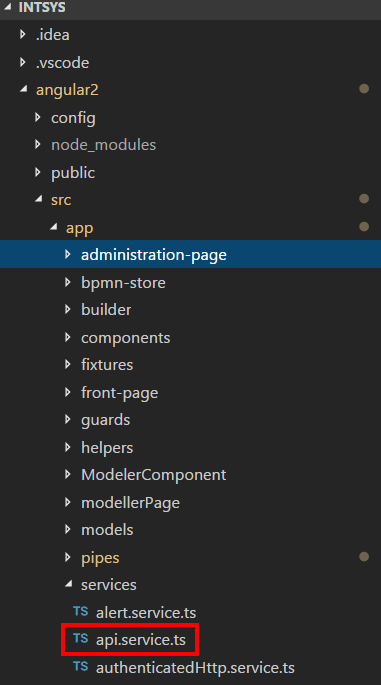


Abbildung 32: api.service.ts verknüpft die Oberfläche mit den Datenbank-Requests

Beispiel für die Dokumentation:

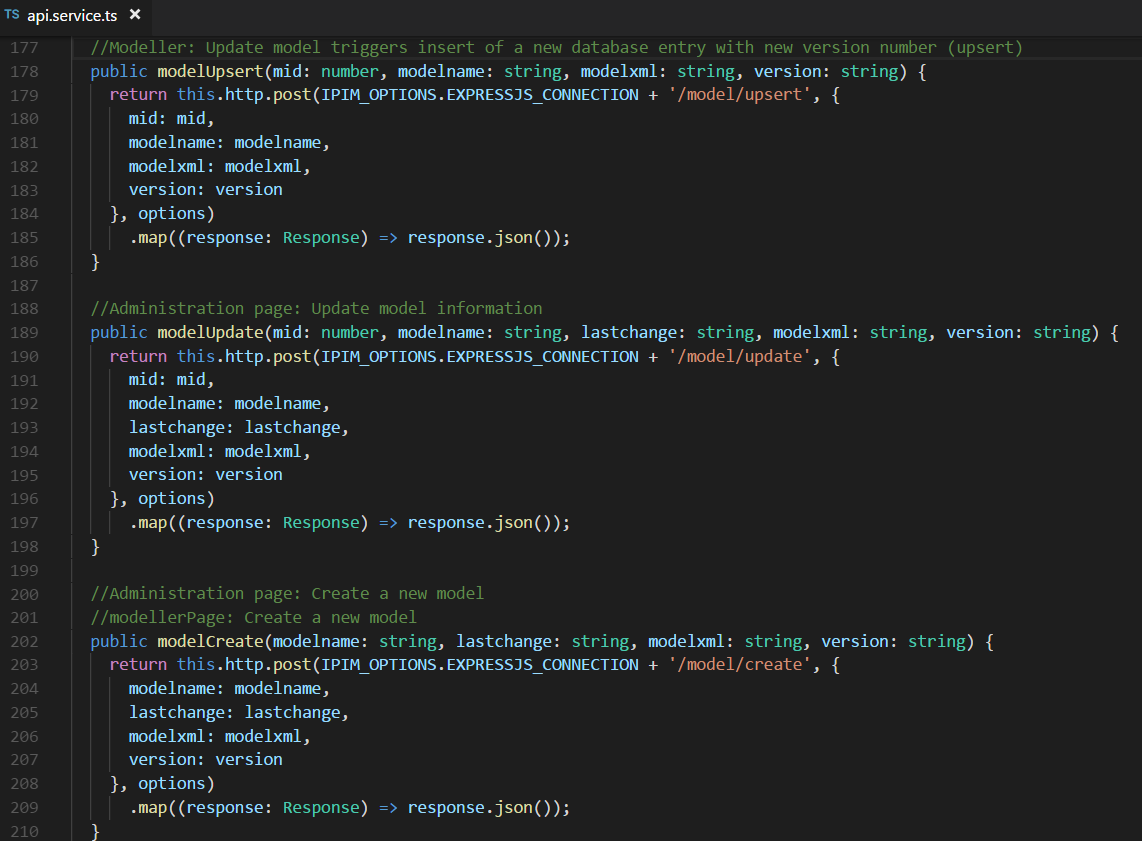


Abbildung 33: Dokumentation der Aufrufe in api.service.ts

# Benutzerhandbuch

## Allgemeiner Aufbau

## Modeller: Modell anlegen, bearbeiten (löschen?)

## Administration Page: Stammdaten anlegen, bearbeiten, löschen

# Arbeitspakete

|  |  |
| --- | --- |
| Name | Tätigkeiten |
| Masood Ahmed |  |
| Jennifer Espich |  |
| Christina Frank |  |
| Granit Gecaj |  |
| Daniel Lackmann |  |
| Markus Schmidtner |  |

# Literaturverzeichnis

Camunda Services GmbH (Hg.) (2018): Modeller. Online verfügbar unter https://camunda.com/products/modeler/, zuletzt aktualisiert am 02.08.2018, zuletzt geprüft am 03.08.2018.