

HOCHSCHULE für Angewandte Wissenschaften LANDSHUT

UNIVERSITY of Applied Sciences

FAKULTÄT Informatik



Projektbericht über das Studienprojekt im WS17/18-SS18

Im Rahmen des Masterstudiengangs Informatik

vorgelegt von

Masood Ahmed

Jennifer Espich

Christina Frank

Granit Gecaj

Daniel Lackmann

Markus Schmidtner

eingereicht am: 07.09.2018

Betreuer:

Daniel Hilpoltsteiner

Prof. Dr. rer. oec. Christian Seel

Inhaltsverzeichnis

Inhaltsverzeichnis	II
Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Abkürzungsverzeichnis	V
1 Allgemeine Beschreibung	1
1.1 Rahmenbedingungen	1
1.2 Entwicklerwerkzeuge	1
2 Technischer Aufbau	3
2.1 Datenbank	3
2.2 MQTT	5
2.3 Express.js	7
2.4 Angular Frontend	7
3 Definition und Umsetzung der User Stories	8
3.1 Umsetzung der User Stories 1 + 2	10
3.2 Umsetzung der User Stories 3 – 6	12
3.3 Umsetzung der User Stories 7 + 8	13
3.4 Umsetzung der User Stories 9 – 10	16
3.5 Umsetzung der User Story 11	17
3.6 Umsetzung der User Story 12	18
3.7 Umsetzung der User Story 13	20
3.8 Umsetzung der User Story 14	21
3.9 Umsetzung der User Story 15	22
4 Installationsvorgehensweise	24
5 Arbeitspakete	25

Abbildungsverzeichnis

Abbildung 1: Datenbanktabellen und ihre Beziehungen zueinander	5
Abbildung 2: Publish/Subscriber Modell bei MQTT	6
Abbildung 3: Starten des Mosca-Servers über die Konsole	6
Abbildung 4: Starten des express.js-Servers über die Konsole	7
Abbildung 5: Modell neu anlegen	10
Abbildung 6: Modell öffnen	11
Abbildung 7: Modell importieren.....	11
Abbildung 8: Modell in der DB speichern	11
Abbildung 9: Kollaboratives Arbeiten an einem Modell	13
Abbildung 10: Subprozess für ein Modell definieren	14
Abbildung 11: Subprozess einem Modell zuordnen.....	15
Abbildung 12: Subprozess in neuem Tab öffnen	15
Abbildung 13: Kaskadierendes Auswerten von Modellen.....	16
Abbildung 14: Anpassung der Werte der Variablen	17
Abbildung 15: Prozessreferenzen anzeigen.....	18
Abbildung 16: Anlage einer Permission.....	19
Abbildung 17: Fehlermeldung ohne Lese-Berechtigungen.....	20
Abbildung 18: Modell als BPMN oder SVG-Image downloaden.....	20
Abbildung 19: Report über Änderungen der letzten 7 Tage	21
Abbildung 20: Anzeige aller User, die das Modell offen haben	22
Abbildung 21: Login mit Anmeldedaten.....	23
Abbildung 22: Fehlgeschlagener Login	23

Tabellenverzeichnis

Tabelle 1: Definition der User Stories	9
Tabelle 2: Umsetzung der User Stories 1 + 2	10
Tabelle 3: Umsetzung der User Stories 3 – 6	12
Tabelle 4: Umsetzung der User Stories 7 + 8	13
Tabelle 5: Umsetzung der User Stories 9 – 10	16
Tabelle 6: Umsetzung der User Story 11	17
Tabelle 7: Umsetzung der User Story 12	18
Tabelle 8: Umsetzung der User Story 13	20
Tabelle 9: Umsetzung der User Story 14	21
Tabelle 10: Umsetzung der User Story 15	22
Tabelle 11: Login-Daten für Testzwecke	23
Tabelle 12: Verteilung der Arbeitspakete	31

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

1 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.

1.1 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.

1.2 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) oder Webstorm realisiert.

Die Präsentationsschicht wurde mit dem Angular 2 Framework 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.

2 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 Benutzungs-schnittstelle.

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.

2.1 Datenbank

Um einen zentralen Speicherpunkt zur Verwaltung der Modelle zu schaffen, 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.

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:

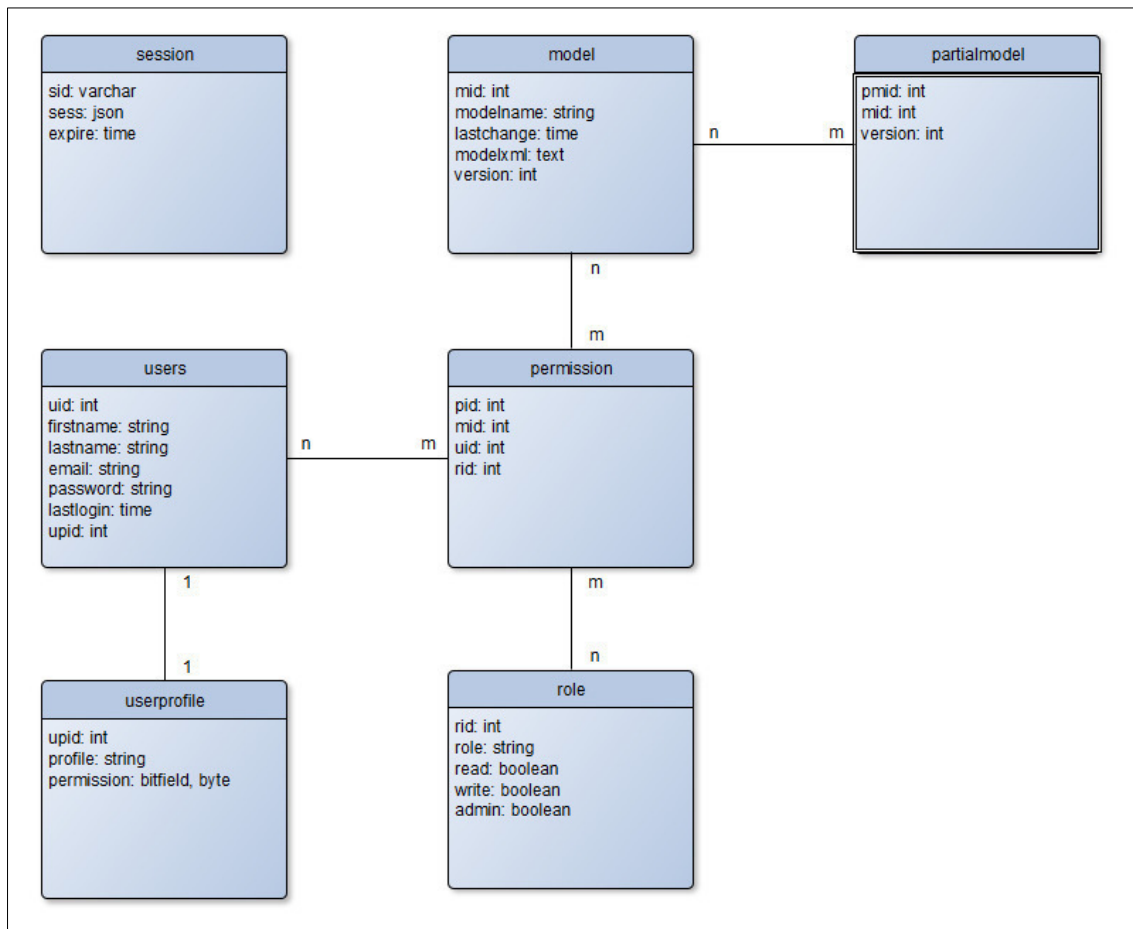


Abbildung 1: Datenbanktabellen und ihre Beziehungen zueinander

2.2 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.

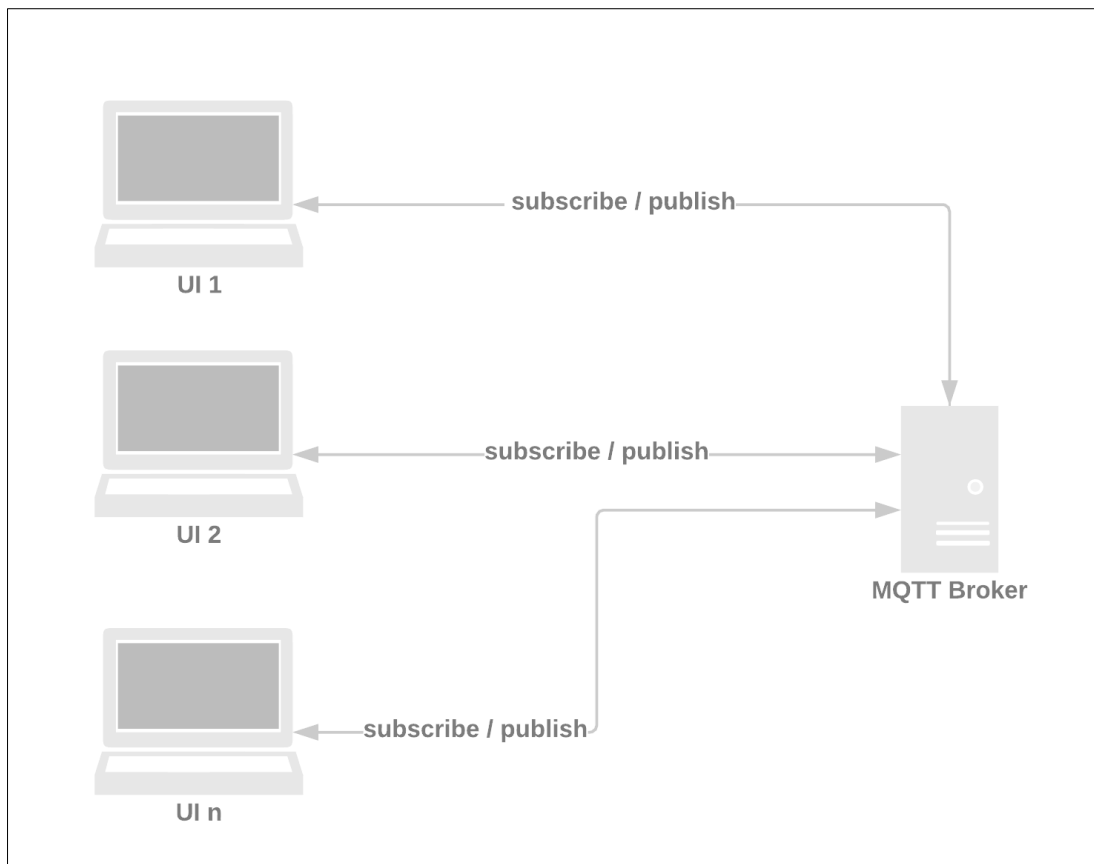


Abbildung 2: Publish/Subscriber Modell bei MQTT

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.

```

PROBLEME  AUSGABE  DEBUGGING-KONSOLE  TERMINAL
2: node

Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\Documents\HAW Landshut\Studienprojekt\IntSys> cd API
PS C:\Users\Documents\HAW Landshut\Studienprojekt\IntSys\API> node .\mqttserver.js
Mosca server is up and running
  
```

Abbildung 3: Starten des Mosca-Servers über die Konsole

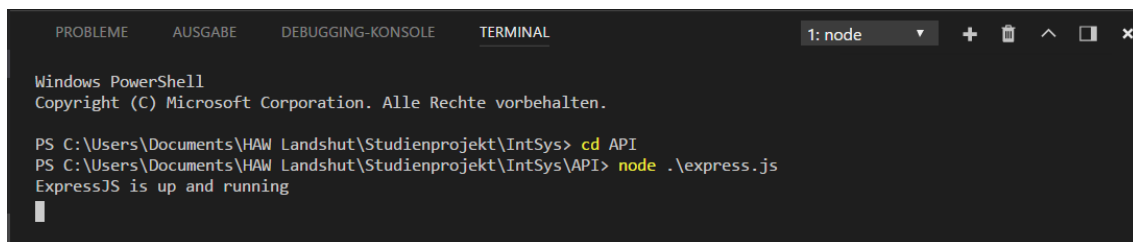
2.3 Express.js

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 wiederum 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.



```

PROBLEME  AUSGABE  DEBUGGING-KONSOLE  TERMINAL
1: node
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\Documents\HAW Landshut\Studienprojekt\IntSys> cd API
PS C:\Users\Documents\HAW Landshut\Studienprojekt\IntSys\API> node .\express.js
ExpressJS is up and running
  
```

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

2.4 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 eingebunden wurde. Die Icons befinden sich in der Palette gezeigt und können unter <https://getbootstrap.com/docs/3.3/components/> für die aktuelle im Projekt verwendete Bootstrap Version 3.3.7 ausgewählt werden.

3 Definition und Umsetzung der User Stories

Zu Beginn des Projekts wurden gemeinsam im Team insgesamt 15 User Stories mit unterschiedlichen Erfüllungsgraden definiert. User Stories mit dem Erfüllungsgrad „Must“ sind Anforderungen, die zwingend in dem Studienprojekt umgesetzt werden müssen. Der Erfüllungsgrad „Should“ besagt, dass diese Anforderungen erfüllt werden sollen. Die höchste Priorität jedoch liegt auf den Anforderungen, die als „Must“ definiert sind. Der letzte Erfüllungsgrad ist „Would“ und beschreibt, dass es schön wäre, diese Anforderungen auch umzusetzen. Anforderungen, die als „Must“ und „Should“ definiert sind haben jedoch höhere Prioritäten.

Tabelle 1 bildet die zu Beginn definierten User Stories ab.

Nr.	Beschreibung	Erfüllungsgrad
1	Als Modellersteller möchte ich ein Diagramm in einer Datenbank speichern können, um es mit anderen zu teilen.	Must
2	Als Modellersteller möchte ich, dass meine Änderung im Diagramm automatisch in die Datenbank übernommen werden, um sie anderen zur Verfügung zu stellen.	Must
3	Als Modellersteller möchte ich, dass Änderungen im Diagramm automatisch an andere User weitergeleitet werden, um sie ihnen zur Verfügung zu stellen.	Must
4	Als Modellersteller möchte ich, dass meine Änderungen mit den Anpassungen anderer zusammengeführt werden, um ein einzelnes Diagramm zu erhalten.	Must
5	Als Modellersteller möchte ich bei Fehlern in der Zusammenführung eine Meldung erhalten, um über weitere Schritte entscheiden zu können.	Must
6	Als Modellersteller möchte ich bei Änderungen von anderen Usern am selbigen Diagramm diese ebenfalls erhalten, um Kenntnis darüber zu haben.	Must

7	Als Modellersteller möchte ich Subprozesse in meinem Diagramm referenzieren können, um bestehende Modelle wiederverwerten zu können	Must
8	Als Modellnutzer möchte ich durch Auswahl eines Subprozesses diesen öffnen können, um einen schnellen Zugriff auf das Diagramm zu haben.	Must
9	Als Modellnutzer möchte ich bei der Auswertung meines Prozesses auch die Variablen des Subprozesses erhalten, um alle Prozesse zeitgleich auswerten zu können.	Should
10	Als Modellnutzer möchte ich, dass bei der Auswertung eines Prozesses automatisch alle Subprozesse ebenfalls ausgewertet werden.	Should
11	Als Modellnutzer möchte ich erkennen, in welchen Diagrammen mein aktuelles Diagramm als Subprozess verwendet wird.	Would
12	Als Modellersteller möchte ich verschiedene Berechtigungs-ebenen für das Modell festlegen können, um die Bearbeitung nur zugelassenen Usern zu erlauben.	Would
13	Als Modellersteller möchte ich gerne eine Übersicht über alle Änderungen der letzten sieben Tage erhalten, um alle Änderung im Überblick zu haben.	Would
14	Als Benutzer möchte ich die Anzahl der aktuell am Dokument arbeitenden User angezeigt bekommen.	Would
15	Als Benutzer möchte ich mich mit Username/Password einloggen, um mich zu identifizieren.	Would

Tabelle 1: Definition der User Stories

3.1 Umsetzung der User Stories 1 + 2

1	Als Modellersteller möchte ich ein Diagramm in einer Datenbank speichern können, um es mit anderen zu teilen.
2	Als Modellersteller möchte ich, dass meine Änderung im Diagramm automatisch in die Datenbank übernommen werden, um sie anderen zur Verfügung zu stellen.

Tabelle 2: Umsetzung der User Stories 1 + 2

Um ein Diagramm in der Datenbank zu speichern, wurden in der model.js entsprechende Endpoints erstellt. Bei der Anlage wird, wenn alle Voraussetzung erfüllt sind, über der „create“ Befehl das Diagramm in der Datenbank gespeichert. Wenn es bearbeitet wurde und erneut gespeichert werden soll, wird der Befehl „upsert“ aufgerufen. Dieser speichert das Modell mit einer neuen Version ab. Neben dem Erzeugen eines neuen, leeren Diagramms, besteht auch die Möglichkeit bestehende .bpmn Dokumente zu importieren, um eine "abwärtskompatibilität" zu gewährleisten.

Auf der Oberfläche wurde für die Speicherung eines Diagramms ein Speicher-Button als Modal eingefügt, s. SaveModal.ts. Bei dessen Betätigung seitens des Endanwenders wird die Methode „saveToDb“ aufgerufen, in der modeler.components.ts definiert ist.

Alle Änderungen an einem Modell werden an zentraler Stelle im express.js zwischengespeichert und an andere User übermittelt, bis das Modell schließlich final in die Datenbank gespeichert wird.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Wenn ein neues Dokument angelegt werden möchte, kann in der Übersichtsseite durch einen Klick auf den Button „New“ ein leeres Dokument geladen werden.

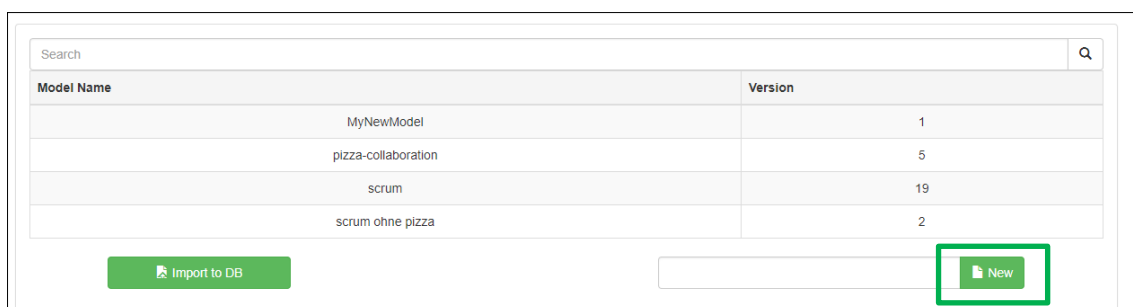


Abbildung 5: Modell neu anlegen

Alternativ kann auch ein bestehendes Modell bearbeitet werden. Hierfür in der Übersichtsseite auf ein Modell klicken, sodass dieses blau markiert wird. Es erscheint dann der Button „Load from DB“. Auf diesen klicken, um das Modell zu öffnen.

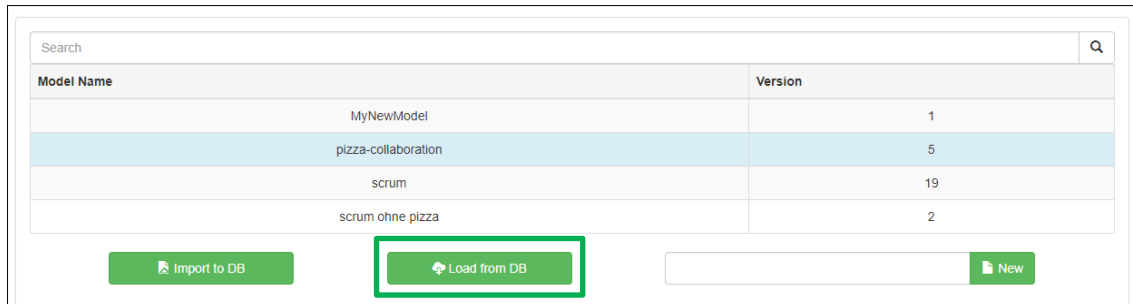


Abbildung 6: Modell öffnen

Eine weitere Möglichkeit, ein Modell zu laden bietet der Button „Import to DB“. Durch diesen können bestehende .bpmn Dokumente in den Modeller importiert werden.

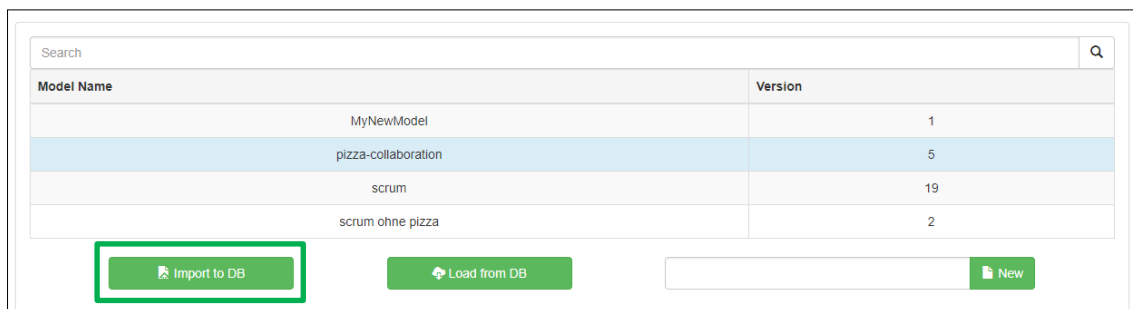


Abbildung 7: Modell importieren

Wenn ein Modell erstellt oder bearbeitet wird, kann das Modell über den Button „Save to Database“ in der linken oberen Ecke im Menü gespeichert werden.

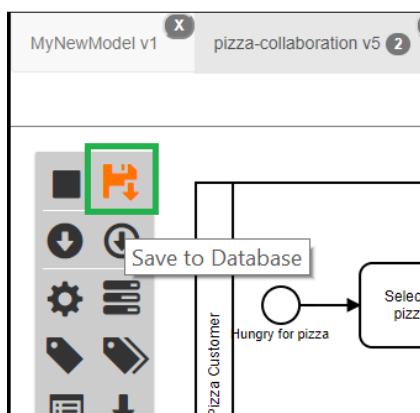


Abbildung 8: Modell in der DB speichern

3.2 Umsetzung der User Stories 3 – 6

3	Als Modellersteller möchte ich, dass Änderungen im Diagramm automatisch an andere User weitergeleitet werden, um sie ihnen zur Verfügung zu stellen.
4	Als Modellersteller möchte ich, dass meine Änderungen mit den Anpassungen anderer zusammengeführt werden, um ein einzelnes Diagramm zu erhalten.
5	Als Modellersteller möchte ich bei Fehlern in der Zusammenführung eine Meldung erhalten, um über weitere Schritte entscheiden zu können.
6	Als Modellersteller möchte ich bei Änderungen von anderen Usern am selbigen Diagramm diese ebenfalls erhalten, um Kenntnis darüber zu haben.

Tabelle 3: Umsetzung der User Stories 3 – 6

Um ein kollaboratives Modellieren zu ermöglichen, wird beim Laden eines Modells aus der Datenbank zuerst eine temporäre Kopie in den express.js geladen. Dieses zur Bearbeitung zur Verfügung stehende Modell wird dann an den User weitergegeben und ein entsprechendes Topic für das Modell im MQTT Server erstellt. Wird nun vom User eine Änderung durchgeführt, wird dies über den Eventbus des Camunda Modelers ermittelt. Daraufhin wird aktuelle Zustand des Modells als XML String an den express.js zurückübermittelt um das temporäre Modell entsprechend anzupassen. Dies passiert ebenfalls über das Topic des MQTT Servers. Wenn nun ein weiterer User dasselbe Modell öffnet, wird ihm stattdessen die temporäre Kopie des express.js geladen und ebenfalls eine Subscription auf das jeweilige Topic ausgeführt. Dies stellt sicher, dass der Anwender von Beginn an das aktuelle Modell erhält und über alle Änderungen informiert wird. Einziger Nachteil dieser Vorgehensweise ist, dass das Modell jeweils komplett übertragen und geladen werden muss. Daher kann es dazu kommen, dass bei zeitgleichem ziehen einer Kante und Ankunft eines aktualisierten Modells das ziehen abgebrochen wird.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Ein Modell kann von beliebig vielen Anwender geöffnet werden. Bei Änderungen eines Anwenders in dem Modell werden die Änderungen in den anderen geöffneten Modellen automatisch nachgezogen.

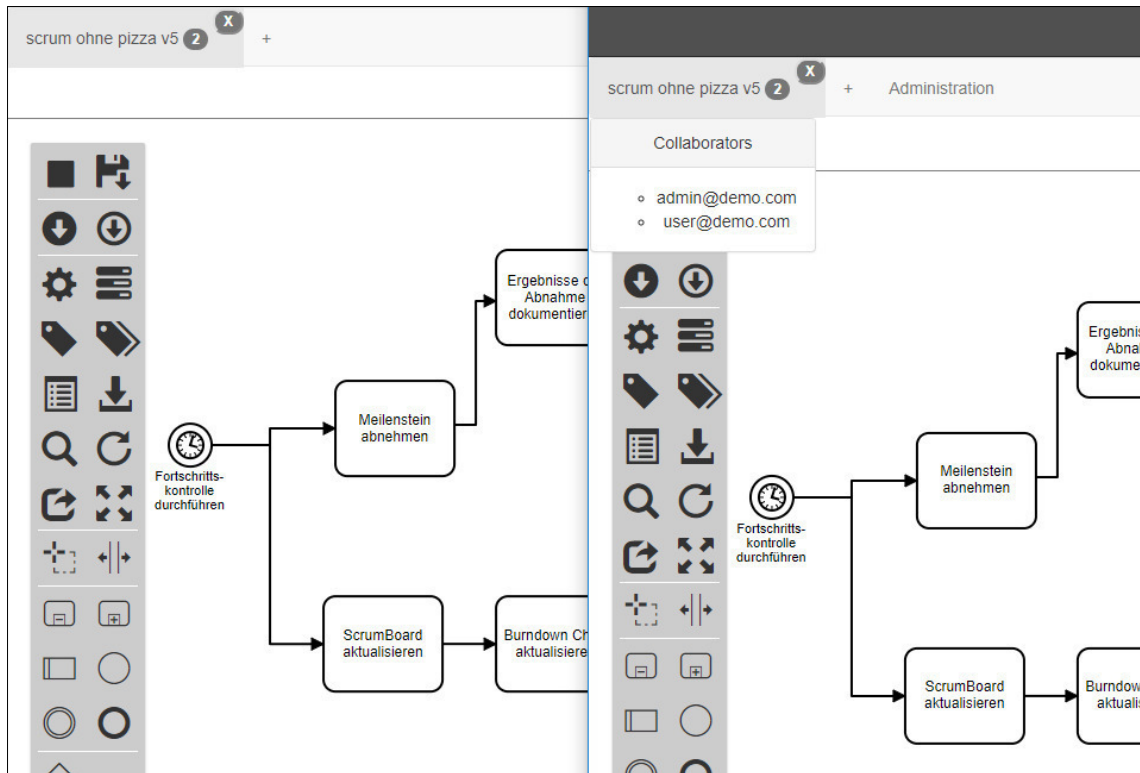


Abbildung 9: Kollaboratives Arbeiten an einem Modell

3.3 Umsetzung der User Stories 7 + 8

7	Als Modellersteller möchte ich Subprozesse in meinem Diagramm referenzieren können, um bestehende Modelle wiederverwerten zu können
8	Als Modellnutzer möchte ich durch Auswahl eines Subprozesses diesen öffnen können, um einen schnellen Zugriff auf das Diagramm zu haben.

Tabelle 4: Umsetzung der User Stories 7 + 8

Diese beiden User Stories wurden im Frontend in der `modeler.component.ts` umgesetzt. Hier gibt es die Methode „`getSubProcessList`“, durch die das Subprozess-Modal geladen wird und alle Subprozesse angezeigt werden, sodass der Anwender sich den entsprechenden Subprozess auswählen kann.

Ein selektierter Subprozess kann mit Hilfe die Methoden „`openSubProcessModel`“ und „`loadSubProcessModel`“ in einem neuen Tab geöffnet werden. Im Backend greift hierfür der GET-Endpoint „`getModel`“.

Praktisch hat dies zur Folge, dass ein User einen selektierten Subprozess Element direkt über ein Icon an der Palette mit einem anderen Diagramm verknüpfen kann. Hierzu öffnet sich ein Menü in dem alle verfügbaren Prozesse als Liste angezeigt werden. Durch simples klicken kann der User nun den gewünschten Prozess auswählen oder ihn auch direkt in einem neuen Tab öffnen.

Des Weiteren ist es möglich, einen selektiertes Subprozess Element auch direkt über ein Icon in der Palette in einem neuen Tab zu öffnen.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Voraussetzung, um diese Funktion an der Oberfläche aufrufen und entsprechend nutzen zu können, ist dass es in dem Modell ein Subprozess gibt. Dieser wird über das kleine „+“ gekennzeichnet.

Wenn ein Subprozess in einem Modell ist und zusätzlich für ein anderes Modell auch gesetzt werden soll, diesen Subprozess anklicken, sodass am Rand durch kleine blaue Striche markiert wird. Erst dann reagiert der Button „Set Subprocess“ im linken Menü.

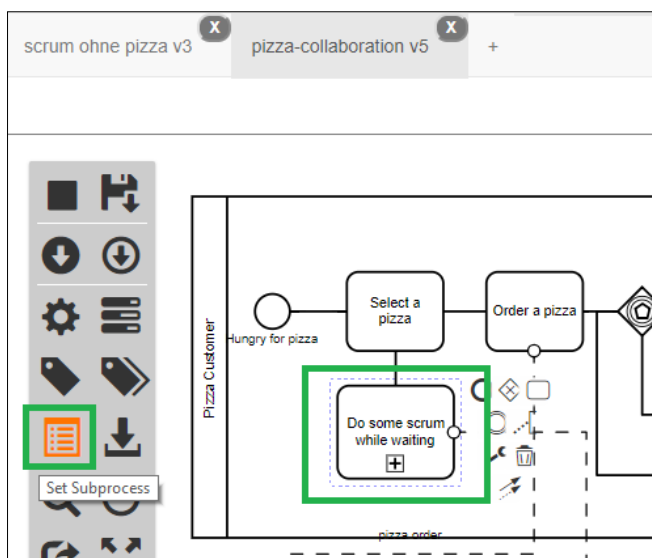


Abbildung 10: Subprozess für ein Modell definieren

In dem sich öffnenden Dialog kann schließlich das Modell ausgewählt werden, dem der Subprozess zugeordnet werden soll. Hierzu ist das entsprechende Modell einfach anzuklicken, sodass es blau markiert wird. Das Modell kann, sofern gewollt, über „Open Modell“ zunächst in einem neuen Tab geöffnet werden, um sicherzustellen, dass das richtige Modell ausgewählt wird. Dieser Schritt ist jedoch optional. Um den Subprozess dem ausgewählten Modell zuzuweisen über den Button „Set Subprocess“ bestätigen.

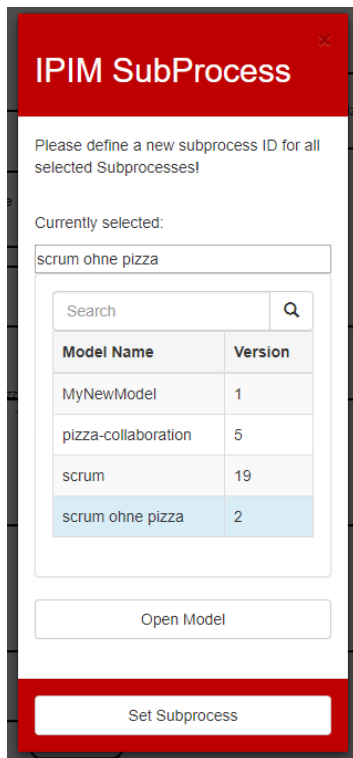


Abbildung 11: Subprozess einem Modell zuordnen

Wenn ein Subprozess in einem Modell ist und als eigenständiges Modell in einem neuen Tab geöffnet werden soll, diesen Subprozess anklicken, sodass am Rand durch kleine blaue Striche gehighlited wird. Erst dann reagiert der Button „Open Model of Subprocess“ im linken Menü und der selektierte Subprozess wird als Modell in einem neuen Tab geladen.

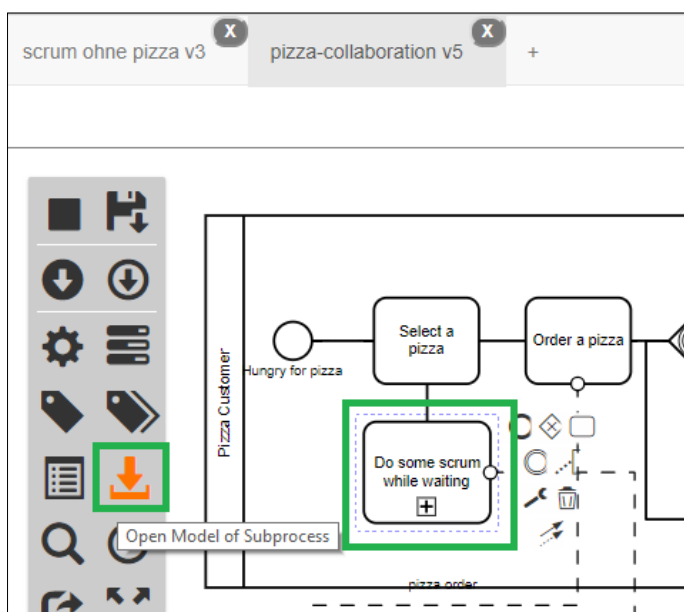


Abbildung 12: Subprozess in neuem Tab öffnen

3.4 Umsetzung der User Stories 9 – 10

9	Als Modellnutzer möchte ich bei der Auswertung meines Prozesses auch die Variablen des Subprozesses erhalten, um alle Prozesse zeitgleich auswerten zu können.
10	Als Modellnutzer möchte ich, dass bei der Auswertung eines Prozesses automatisch alle Subprozesse ebenfalls ausgewertet werden.

Tabelle 5: Umsetzung der User Stories 9 – 10

Um diesen User Stories gerecht zu werden, wurde ein Evaluator für kaskadierendes Auswerten geschrieben. Wird dieser über die Palette für ein Modell gestartet, werden zunächst sämtliche referenzierten Subprozesse ermittelt und aus der Datenbank nachgeladen. Sobald diese Verfügbar sind, werden diese ebenfalls auf weitere referenzierte Subprozesse geprüft. Hierbei wird jeder Prozess nur einmal nachgeladen, sodass ein Zirkel Bezug ausgeschlossen ist. Sind alle Subprozesse geladen, werden alle so bezogenen Modelle auf hinterlegte Variablen ausgewertet. Diese werden kombiniert, um Redundanzen auszuschließen, und schließlich in einem Modal dem User zur Bearbeitung präsentiert. Sobald der Anwender die Daten angepasst hat, kann er die Auswertung der Modelle starten, die dann entsprechend der Variablen adaptiert werden. Ist auch dieser Teil abgeschlossen, werden alle Modelle als .zip Datei zum Download bereitgestellt.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Für das kaskadierende Auswerten muss das Modell offen sein. Hierfür im linken Menü den Button „Start cascading Evaluation“ drücken, um die Auswertung zu beginnen.

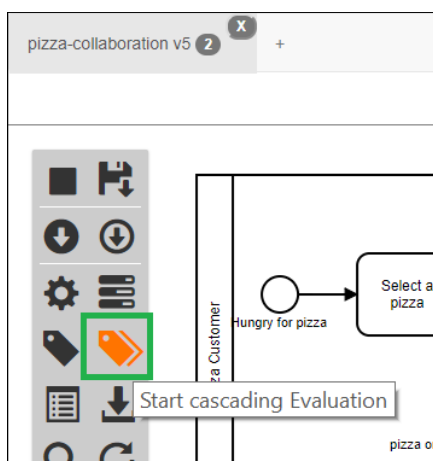


Abbildung 13: Kaskadierendes Auswerten von Modellen

Im folgenden Dialog können die Werte der zuvor festgelegten Variablen nach Belieben angepasst werden. Die Default-Werte aus dem Dialog zum Definieren von Variablen sind hier bereits voreingetragen. Meta-Variablen werden in diesem Dialog nicht mehr angezeigt. Sind alle Variablen auf den gewünschten Wert gesetzt, kann die Auswertung über den Button „Evaluate“ gestartet werden. Das Modell und die Teilmodelle werden dabei als ZIP-Datei lokal gespeichert.

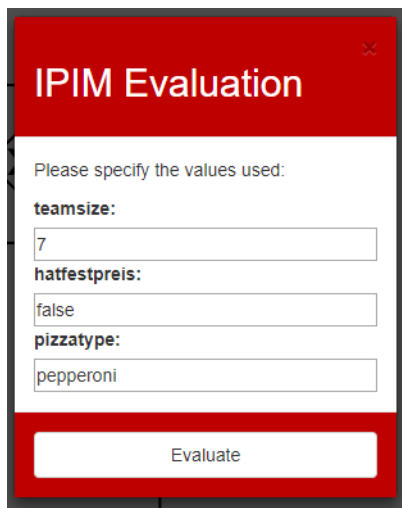


Abbildung 14: Anpassung der Werte der Variablen

3.5 Umsetzung der User Story 11

11	Als Modellnutzer möchte ich erkennen, in welchen Diagrammen mein aktuelles Diagramm als Subprozess verwendet wird.
----	--

Tabelle 6: Umsetzung der User Story 11

Da ein Modell von mehreren Modellen als Subprozess verwendet werden, können mit Hilfe die Methode „returnSubProcessList“, die in der `modeler.component.ts` definiert ist, die Modelle anzeigen lassen, in dem das Modell als Subprozess verwendet wird.

Hierzu wird in der Datenbank eine eigene Tabelle geführt, in der hinterlegt ist welche Diagramme von jeweils anderen referenziert werden. Diese Information wird jeweils beim Speichern eines Modells aktualisiert und referenziert immer die neueste Version eines Diagramms. Hierdurch ist bei einer Änderung keine Anpassung der referenzierenden Diagramme notwendig. Durch ein simples Icon in der Palette kann der User ein Modal einblenden, welches die entsprechenden Datenbankeinträge ausliest und so dem User anzeigt welche anderen Modell auf den jeweiligen Tab referenzieren.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Um den Überblick zu behalten, ob ein Modell auch ein Subprozess ist und von anderen Modellen verwendet wird kann die Funktion „See Process references“ im linken Menü herangezogen werden. Der sich öffnende Dialog gibt an, in welchem/n Modell/en das selektierte Modell als Subprozess verwendet wird.

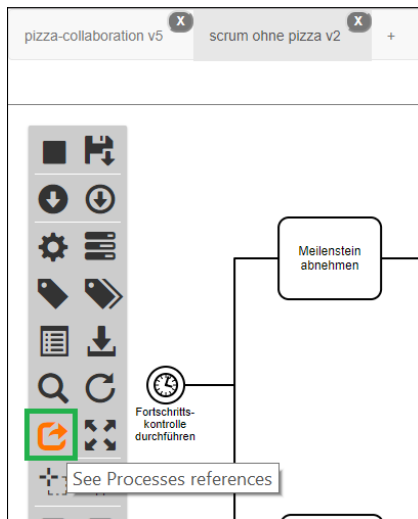


Abbildung 15: Prozessreferenzen anzeigen

3.6 Umsetzung der User Story 12

12	Als Modellersteller möchte ich verschiedene Berechtigungsebenen für das Modell festlegen können, um die Bearbeitung nur zugelassenen Usern zu erlauben.
----	---

Tabelle 7: Umsetzung der User Story 12

Die Logik für diese Anforderungen wurde einerseits in der role.js und in der permission.js definiert. Die role.js beinhaltet die Endpoints zum Erstellen und Anpassen einer Rolle. Einer Rolle werden unterschiedliche Rechte (read, write, admin) zugewiesen. In der permission.js wird eine Rolle mit einem User und einem Model zusammengefügt, um somit einem Anwender auf einem Model Berechtigungen zu geben. Standardmäßig erhält ein neuer User für alle Modell read und write.

Der User erhält bereits im Dialog zum Laden eines Modells eine Übersicht über die jeweiligen Berechtigungen die er auf das Modell hat. Dies wird durch entsprechende Icons repräsentiert. Verfügt ein User über kein read recht, kann er das Modell nicht öffnen und eine Fehlermeldung erscheint. Mit nur der Read Berechtigung, lässt sich zwar

das Modell öffnen, es lässt sich jedoch nicht zurück in die Datenbank speichern. Ein Download als SVG oder BPMN ist aber immer noch möglich. Icons im jeweiligen Diagramm Tab zeigen jeweils die Berechtigungen an.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Die Berechtigung für ein Modell muss von einem Administrator angelegt werden. Um eine neue Berechtigung anzulegen, den Anwender, der eine Berechtigung benötigt auswählen, sodass dieser blau markiert wird. Im Anschluss auf das entsprechende Model navigieren, auf dass der Anwender die Berechtigung benötigt. Wenn sowohl Anwender und Model markiert sind, erscheint unten im Bereich die Rolle, mit der der User Rechte auf das gewählte Modell erhält. Über den Button „Change relation“ wird die Berechtigung angelegt bzw. geändert. Der Admin bekommt entsprechend eine Erfolgsmeldung.

User	Model	Role	Permission											
Permission														
<table border="1"> <thead> <tr> <th>Email</th> </tr> </thead> <tbody> <tr><td>user@demo.com</td></tr> <tr><td>masood@md.de</td></tr> <tr><td>Markus@haw-landshut.de</td></tr> <tr><td>lackmann1994@gmail.com</td></tr> <tr><td>admin@demo.com</td></tr> </tbody> </table>	Email	user@demo.com	masood@md.de	Markus@haw-landshut.de	lackmann1994@gmail.com	admin@demo.com	<table border="1"> <thead> <tr> <th>ModelName</th> </tr> </thead> <tbody> <tr><td>MyNewModel</td></tr> <tr><td>pizza-collaboration</td></tr> <tr><td>scrum</td></tr> <tr><td>scrum ohne pizza</td></tr> </tbody> </table>	ModelName	MyNewModel	pizza-collaboration	scrum	scrum ohne pizza		
Email														
user@demo.com														
masood@md.de														
Markus@haw-landshut.de														
lackmann1994@gmail.com														
admin@demo.com														
ModelName														
MyNewModel														
pizza-collaboration														
scrum														
scrum ohne pizza														
Markus@haw-landshut.de - pizza-collaboration 5														
<table border="1"> <thead> <tr> <th>Role</th> </tr> </thead> <tbody> <tr> <td>ReadOnly</td> </tr> </tbody> </table>		Role	ReadOnly	<div> <div>change relation</div> <div>delete</div> </div>										
Role														
ReadOnly														

Abbildung 16: Anlage einer Permission

Wenn ein Anwender auf ein Modell keine Lese-Berechtigungen hat, erhält er entsprechend eine Fehlermeldung.

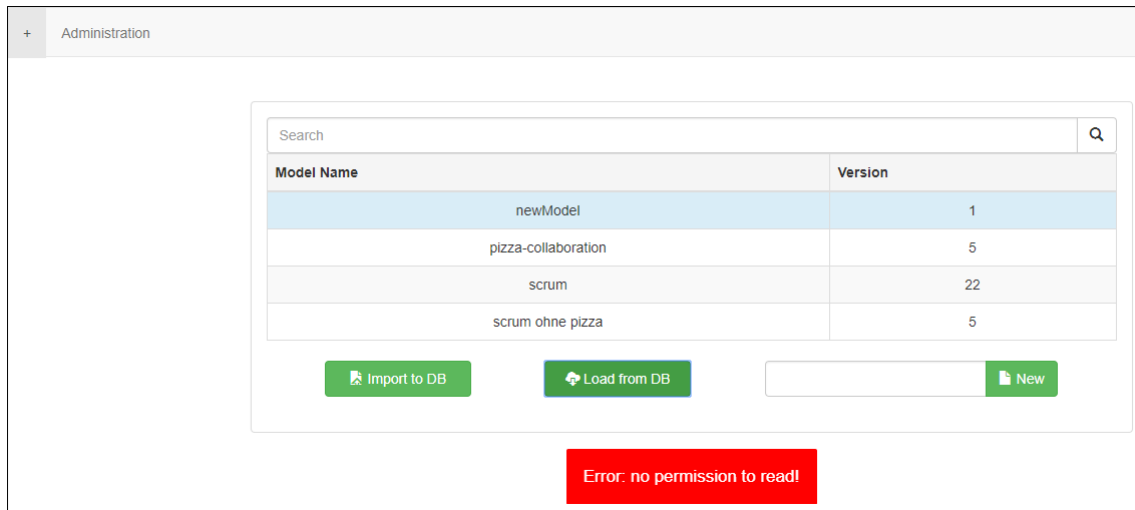


Abbildung 17: Fehlermeldung ohne Lese-Berechtigungen

Wenn ein Anwender auf ein Modell zwar Lese-Berechtigungen hat, jedoch keine Schreib-Rechte, kann er das Modell öffnen, jedoch Änderungen, die er vornimmt nicht speichern. Hingegen kann er das geänderte Modell über die folgenden Buttons als SVG- oder BPMN-Datei herunterladen.

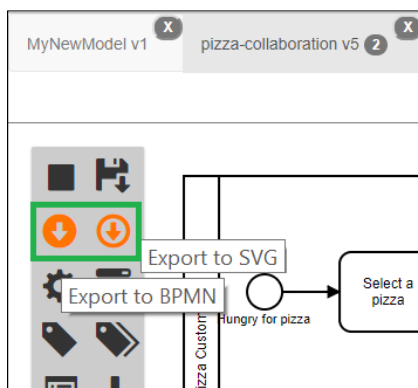


Abbildung 18: Modell als BPMN oder SVG-Image downloaden

3.7 Umsetzung der User Story 13

13	Als Modellersteller möchte ich gerne eine Übersicht über alle Änderungen der letzten sieben Tage erhalten, um alle Änderung im Überblick zu haben.
----	--

Tabelle 8: Umsetzung der User Story 13

Um die Modelle anzuzeigen, die in den letzten sieben Tagen bearbeitet wurden, wurde im Frontend die Methode „getLatestChanges“ in der modelloader.component.ts erstellt.

Durch diese wird über die `api.service.ts` der GET-Endpoint „changes“ in der `model.ts` aufgerufen. Hier ist definiert, dass die Anzeige der Modelle hinsichtlich des Datums absteigend sortiert werden soll, sodass das Model, das zuletzt geändert wurde, als erstes angezeigt wird.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Der Report, der die Modelle anzeigt, die in den letzten sieben Tagen geändert wurden, findet sich nach dem Login auf der Übersichtsseite. Die geänderten Modelle werden hinsichtlich des Bearbeitungsdatums absteigend sortiert, sodass die neueste Änderung stets ganz oben aufgeführt ist.

Latest Changes:		
Model Name	Version	Last Change
newModel	1	31.08.2018 15:02
scrum	21.1	31.08.2018 14:47
scrum	22	31.08.2018 14:46
scrum ohne pizza	5	31.08.2018 14:29
scrum ohne pizza	4	31.08.2018 14:29
scrum ohne pizza	2.3.1	31.08.2018 13:05
scrum ohne pizza	2.4	31.08.2018 13:05
scrum ohne pizza	2.3	31.08.2018 13:05

Abbildung 19: Report über Änderungen der letzten 7 Tage

3.8 Umsetzung der User Story 14

14	Als Benutzer möchte ich die Anzahl der aktuell am Dokument arbeitenden User angezeigt bekommen.
----	---

Tabelle 9: Umsetzung der User Story 14

Der Express.js Server fungiert auch für diese User Story als zentrale Einheit. Er speichert nicht nur das temporäre Modell, sondern wird zudem vom MQTT Server über jeweils neue Subscriptions auf das Topic des Modells informiert. Dem Express.js Server ist also die Gesamtzahl der Subscriber (aktuell geöffnete Modelle von Usern) bekannt. Dieser Wert wird ebenfalls per MQTT Message an die Clients übertragen und so eine

Anzeige ermöglicht. Diese erfolgt direkt im Tab des Diagramm, direkt hinter dem Namen. Ist kein User mehr auf ein Model Subscribed wird das temporäre Diagramm verworfen.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Beim kollaborativen Modellieren werden in der Navigationsbar neben dem Modellnamen die Anzahl der Anwender angezeigt, die ein Modell gleichzeitig geöffnet haben (hier 2). Bei Navigation auf diese Zahl werden die E-Mail-Adressen der Anwender angezeigt.

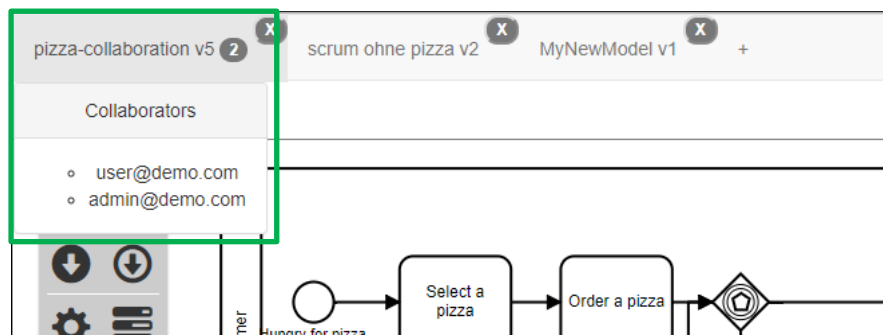


Abbildung 20: Anzeige aller User, die das Modell offen haben

3.9 Umsetzung der User Story 15

15	Als Benutzer möchte ich mich mit Username/Password einloggen, um mich zu identifizieren.
----	--

Tabelle 10: Umsetzung der User Story 15

Die „login“ Methode der GUI, bei der der Endanwender sich einloggen muss, befindet sich in der front-page.component.ts. Im Backend wird in der express.js der POST-Endpoint „authenticate“ aufgerufen, der dem Anwender eine gültige Session zuweist.

Die Umsetzung an der Oberfläche sieht demnach wie folgt aus:

Der Anwender kann sich mit seiner E-Mail und das zugehörige Passwort einloggen.



Abbildung 21: Login mit Anmeldedaten

Voraussetzung, dass der Anwender sich einloggen kann ist, dass ein Administrator den User im Modeller bereits angelegt hat. Der Anwender erhält im besten Fall die Login-Daten seitens des Admins. Wenn der Anwender noch nicht angelegt wurde erhält er eine Fehlermeldung.



Abbildung 22: Fehlgeschlagener Login

Login-Daten für Testzwecke:

	Ohne Adminrechte	Mit Adminrechten
User	user@demo.com	admin@demo.com
Passwort	12341234	12341234

Tabelle 11: Login-Daten für Testzwecke

4 Installationsvorgehensweise

➤ Schritt 1

Github Account anlegen

➤ Schritt 2:

Git installieren und Repo lokal clonen

➤ Schritt 3:

VS Code auf jedem Rechner installieren

➤ Schritt 4:

Node.js installieren

➤ Schritt 5:

*lint Komponenten in VS Code installieren

➤ Schritt 6:

PostgreSQL und pgAdmin installieren

➤ Schritt 7:

Open VPN installieren, um remote zu arbeiten

5 Arbeitspakete

Aufgaben	Bearbeiter
Entwicklungsumgebung	
VSCode/IntelliJ installieren	alle
NPM installieren	alle
VCS einrichten	alle
Dependencies anpassen	Masood Ahmed, Markus Schmidtner
Initiale Fehler beheben	alle
User Login	
Login Page erstellen	Daniel Lackmann
Login Page Design	Masood Ahmed, Granit Gecaj
Mit Sessionmanagement verknüpfen	Daniel Lackmann
Kennwort Verschlüsselung	Daniel Lackmann
Sessionmanagement	
Sessionmanagement in Backend implementieren	Daniel Lackmann
Sessionmanagement Frontend implementieren	Daniel Lackmann
Auth-Gard implementieren	Daniel Lackmann

Modeler Integration	
Variablen Modal einfügen	Markus Schmidtnr, Christina Frank
Variablen Modal Obeflächengestaltung	Christina Frank, Markus Schmidtnr
Term Modal einfügen	Markus Schmidtnr, Christina Frank
Term Modal Obeflächengestaltung	Christina Frank, Markus Schmidtnr
Subprocess Modal einfügen	Markus Schmidtnr
Subprocess Modal Obeflächengestaltung	Christina Frank, Markus Schmidtnr
Subprocess Modal mit Loading Komponente verbinden	Markus Schmidtnr
Subprozesse aus Modal laden	Markus Schmidtnr
Input Modal einfügen	Markus Schmidtnr, Christina Frank
Input Modal Obeflächengestaltung	Christina Frank, Markus Schmidtnr
Selektierten Subprozess öffnen	Markus Schmidtnr
Save Modal einfügen	Daniel Lackmann
Useage Modal einfügen	Markus Schmidtnr
Useage Modal Obeflächengestaltung	Markus Schmidtnr
Evaluator Modal einfügen	Markus Schmidtnr
Evaluator Modal Obeflächengestaltung	Christina Frank, Markus Schmidtnr

Model als SVG speichern	Markus Schmidtner
Model als BPMN speichern	Markus Schmidtner
Highlight Terms	Markus Schmidtner
Reset Diagram	Markus Schmidtner
Zoom to Fit Funktion	Markus Schmidtner
Ursprungskoordinaten im Modelerview	Markus Schmidtner
Übersetzung mit ngTranslate	Markus Schmidtner
Auswertung mit SafeEval	Markus Schmidtner
Mehrere Modeler in Tabs organisieren	Daniel Lackmann
Model aus Datenbank laden	Daniel Lackmann
Model in Datenbank speichern	Daniel Lackmann
Model von Dateisystem importieren	Markus Schmidtner, Daniel Lackmann
Leeres Model in Datenbank erzeugen	Markus Schmidtner, Daniel Lackmann
Icon Palette erweitern	Markus Schmidtner
Overlay Texte und Hover Effekte hinzufügen	Markus Schmidtner
IPIM Logo im Modeler einfügen	Masood Ahmed, Markus Schmidtner

Oberfläche Administratoren	
Funktionalität für User Verwaltung	Daniel Lackmann
Funktionalität für Model/Permission Verwaltung	Daniel Lackmann, Markus Schmidtner
Funktionalität für Profil Verwaltung	Daniel Lackmann
Funktionalität für Rollenverwaltung	Daniel Lackmann
CSS Design für User Verwaltung	Granit Gecaj, Masood Ahmed
CSS Design für Model/Permission Verwaltung	Granit Gecaj, Masood Ahmed
CSS Design für Profil Verwaltung	Granit Gecaj, Masood Ahmed
CSS Design für Rollenverwaltung	Granit Gecaj, Masood Ahmed
Suchfunktion für Administration	Daniel Lackmann
Email Validation	Masood Ahmed
Model Merging	
Command Stack	Markus Schmidtner
Command Interceptor	Markus Schmidtner
Event Bus abfangen	Markus Schmidtner
MQTT/MOSCA Server aufsetzen	Markus Schmidtner
MQTT Client in Command Interceptor integrieren	Markus Schmidtner

Abort Action vor Import um Modeler Status zurückzusetzen vor Import	Markus Schmidtner
ExpressJS Speicherung von bearbeiteten Modellen	Daniel Lackmann
Anzahl der Subscriber eines Models im Tab anzeigen	Daniel Lackmann
Datenbank erstellen	
Datenbank Server aufsetzen	Markus Schmidtner
Postgres Datenbank aufsetzen	Markus Schmidtner
Datenbank Schema definieren	alle
Datenbanktabellen erzeugen	Jennifer Espich
ExpressJS/Datenbank abfragen	
ExpressJS aufsetzen	Markus Schmidtner
Postman einrichten	Jennifer Espich, Markus Schmidtner
Abfragen für User Tabelle erstellen	Jennifer Espich, Daniel Lackmann
Abfragen für User Tabelle mit Postman testen	Jennifer Espich
Abfragen für Model Tabelle erstellen	Jennifer Espich, Daniel Lackmann
Abfragen für Model Tabelle mit Postman testen	Jennifer Espich
Abfragen für Teilmodell Tabelle erstellen	Jennifer Espich, Markus Schmidtner
Abfragen für Teilmodell Tabelle mit Postman testen	Jennifer Espich

Abfragen für Permission Tabelle erstellen	Jennifer Espich
Abfragen für Permission Tabelle mit Postman testen	Jennifer Espich
Abfragen für Profil Tabelle erstellen	Jennifer Espich
Abfragen für Profil Tabelle mit Postman testen	Jennifer Espich
Abfragen für Rollen Tabelle erstellen	Jennifer Espich
Abfragen für Rollen Tabelle mit Postman testen	Jennifer Espich
API Service für ExpressJS Abfragen	Daniel Lackmann
Alert Service für Fehlermeldungen	Daniel Lackmann
Kaskadirendes Auswerten	
Hintergrundauswertung von Modellen	Markus Schmidtner
Abfrage von verknüpften Subprozessen	Markus Schmidtner
rekursives Nachladen und Auswerten von Subprozessen	Markus Schmidtner
Ergebnisse zippen und bereitstellen	Markus Schmidtner
Asynchrone Prozessbearbeitung	Markus Schmidtner

Tests	
Aufsetzen von jasmine/karma	Markus Schmidtner
Simple Basis Tests	Markus Schmidtner
Fake-Backend für Tests	Markus Schmidtner
API Beispieltests	Markus Schmidtner
Kopie Tests für Komponenten/API	Masood Ahmed
Sonstiges	
Ladeanimation für Angular	Markus Schmidtner
Lade Overlay für Wartezeiten	Markus Schmidtner
Änderung der letzten x Tage	Markus Schmidtner
Snackbar/Toast für Modeler Events	Markus Schmidtner
Benachrichtigungsservice	Markus Schmidtner
Darstellung von Permissions über Icons	Markus Schmidtner
Eingeloggter Username mit Bild und Menü in Statusleiste	Markus Schmidtner
Developer Guide für Entwickler	Christina Frank, Jennifer Espich, Masood Ahmed
Benutzerhandbuch für Anwender	Jennifer Espich
API Doc für expressJS	Daniel Lackmann, Jennifer Espich

Tabelle 12: Verteilung der Arbeitspakete