**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](#_Toc523996418)

[Abbildungsverzeichnis III](#_Toc523996419)

[Tabellenverzeichnis IV](#_Toc523996420)

[Abkürzungsverzeichnis V](#_Toc523996421)

[1 Allgemeine Beschreibung 1](#_Toc523996422)

[1.1 Rahmenbedingungen 1](#_Toc523996423)

[1.2 Entwicklerwerkzeuge 1](#_Toc523996424)

[2 Technischer Aufbau 3](#_Toc523996425)

[2.1 Datenbank 3](#_Toc523996426)

[2.2 MQTT 5](#_Toc523996427)

[2.3 Express.js 7](#_Toc523996428)

[2.4 Angular Frontend 7](#_Toc523996429)

[3 Definition und Umsetzung der User Stories 8](#_Toc523996430)

[3.1 Übersicht aller User Stories 9](#_Toc523996431)

[4 Installationsanleitung 10](#_Toc523996432)

[5 Arbeitspakete 11](#_Toc523996433)

# Abbildungsverzeichnis

[Abbildung 1: Datenbanktabellen und ihre Beziehungen zueinander 5](#_Toc523996409)

[Abbildung 2: Publish/Subscriber Modell bei MQTT 6](#_Toc523996410)

[Abbildung 3: Starten des Mosca-Servers über die Konsole 6](#_Toc523996411)

[Abbildung 4: Starten des express.js-Servers über die Konsole 7](#_Toc523996412)

# Tabellenverzeichnis

[Tabelle 1: Definition der User Stories 9](#_Toc523996414)

# 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 Funktio,n 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) 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.

# 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

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:

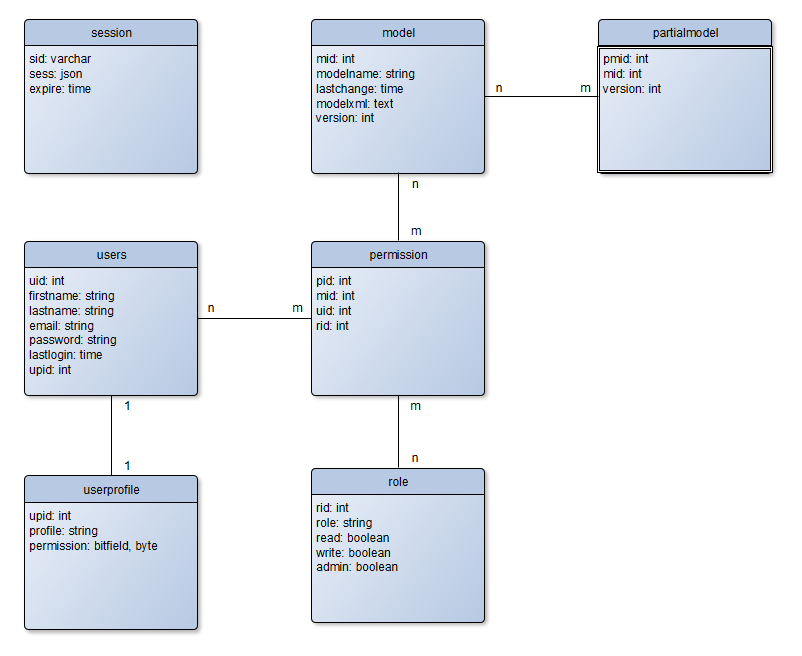


Abbildung 1: Datenbanktabellen und ihre Beziehungen zueinander

## 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.

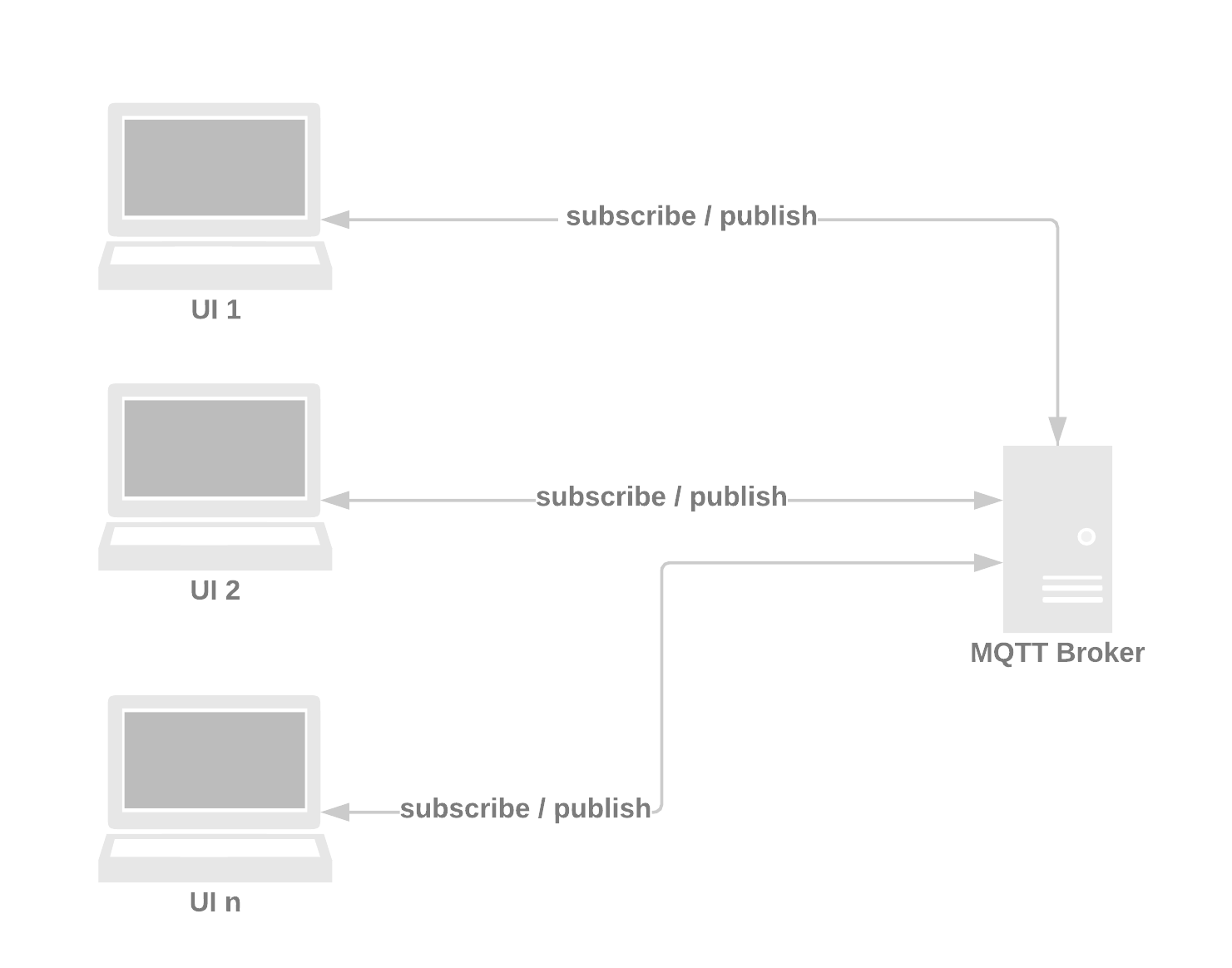


Abbildung 2: Publish/Subscriber Modell bei MQTT

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.

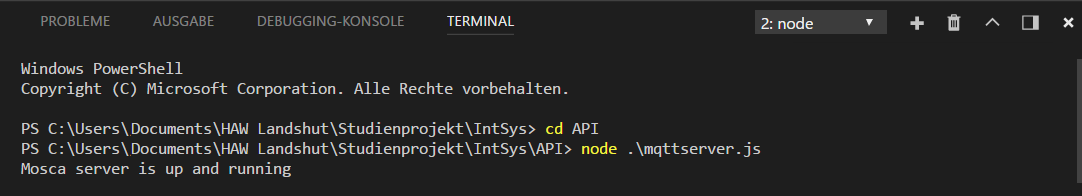


Abbildung 3: Starten des Mosca-Servers über die Konsole

## 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.

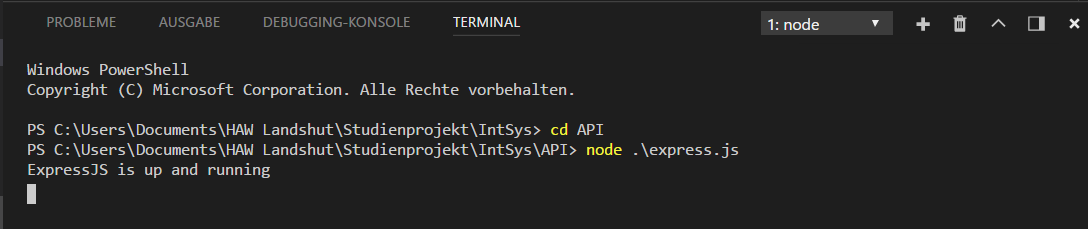


Abbildung 4: Starten des express.js-Servers über die Konsole

## 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 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.

# 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 Berechtigungsebenen 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

## 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.

## 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. |

Umsetzung der User Stories 3 – 6

Um eine 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 dass ziehen abgebrochen wird.

## 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. |

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.

## 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 3: 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 entsprechend angepasst hat, kann er die Auswertung der Modelle starten, die dann entsprechend der Variablen adaptiert werden. Ist auch dieser Teil abgeschlossen, werden alle Modell als .zip Datei zum Download bereitgestellt.

## Umsetzung der User Story 11

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

Tabelle 4: 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.

## 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 5: 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 Diagram Tab zeigen jeweils die Berechtigungen an.

## 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 6: 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.

## Umsetzung der User Story 14

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

Tabelle 7: 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.

## Umsetzung der User Story 15

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

Tabelle 8: 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.

# 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

# Arbeitspakete

s. separate Words-Datei wird dann als PDF (Querformat) eingefügt