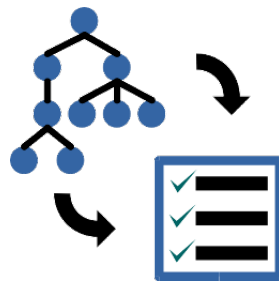


EEPPi



Entwurfsentscheidungen als Projektplanungsinstrument

Technischer Bericht

Abteilung Informatik
Hochschule für Technik Rapperswil
Bachelorarbeit, Herbstsemester 2014

Autoren: Laurin Murer, Tobias Blaser
Betreuer: Prof. Dr. Olaf Zimmermann
Projektpartner: IFS, HSR
Experte: Dr. Gerald Reif, Innovation Process Technology AG
Gegenleser: Prof. Hans Rudin
Abgabedatum: 19. Dezember 2014

1. Abstract

«Entwurfsentscheidungen als Projektplanungsinstrument», kurz EEPPI. Diesem Thema widmet sich die vorliegende Arbeit und befasst sich mit der Frage, ob sich aus Projektentscheidungen Aufgaben ableiten lassen. Weiterhin wird untersucht, ob sich dieser Prozess automatisieren lässt.

Jedes Projekt erfordert das Treffen von Entscheidungen, wobei aus einer bestimmten Entscheidung häufig ähnliche Aufgaben resultieren. Sowohl auf Seite der Entscheidungsverwaltung wie auf Seiten der Projektplanung existieren bereits verschiedene Werkzeuge. Ziel von EEPPI ist es, eine Brücke zwischen Entscheidungsmanagement und Projektplanung zu bauen.

Im Rahmen der Arbeit wurde eine Webapplikation entwickelt, die mögliche Entscheidungen aus einem angebundenen Wissensverwaltungssystem bezieht und dem Benutzer mit einem Metamapping ermöglicht, Projektentscheidungen mit eigenen Aufgaben zu verknüpfen. Die Applikation bietet ihm auch eine weitgehende Konfiguration der angebundenen Systeme und der Aufgabenerzeugung. Ein dazu entwickelter Templatingmechanismus ermöglicht ihm, eigene Verarbeitungsfunktionen, sogenannten Prozessoren, zu verwenden.

EEPPI zeigt, was kommerzielle Produkte in diesem Bereich anbieten könnten, aber auch die Design-Herausforderungen einer solchen Software: Hohe Flexibilität und Konfigurierbarkeit. EEPPI legt somit einen wichtigen Meilenstein im Forschungsbereich des interdisziplinären Entscheidungs- und Projektmanagements und zeigt einen möglichen Entwicklungspfad für zukünftige Tools auf.

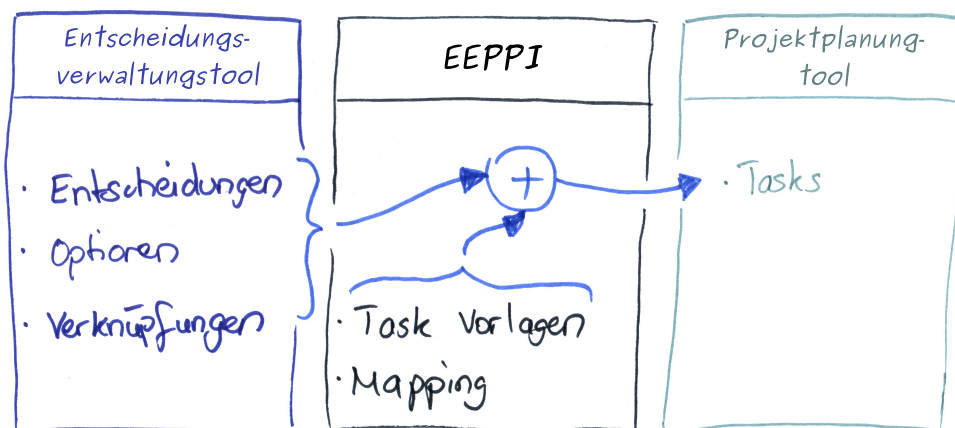
2. Management Summary

2.1. Ausgangslage

«Entwurfsentscheidungen als Projektplanungsinstrument», kurz EEPPI. Diesem Thema widmet sich die vorliegende Arbeit und befasst sich mit der Frage, ob sich aus Projektentscheidungen Aufgaben ableiten lassen. Weiterhin wird untersucht, ob sich dieser Prozess automatisieren lässt.

Jedes Projekt erfordert das Treffen von Entscheidungen. So führt die Entscheidung «Welche Art Session State soll verwendet werden?» zum Beispiel zu den Aufgaben «Session State evaluieren» und «Prototyp umsetzen». Wird bei dieser Entscheidung die Option «Database Session State» ausgewählt, so resultieren aus diesem Entscheid beispielsweise die Aufgaben «Datenbank installieren» und «Session Persistenz implementieren».

Sowohl auf Seite der Entscheidungsverwaltung wie auf Seiten der Projektplanung existieren bereits verschiedene Werkzeuge. Ziel von EEPPI ist es, eine Brücke zwischen Entscheidungsmanagement und Projektplanung zu bauen.



EEPPI bildet eine Brücke zwischen Entscheidungsmanagement- und Projektplanung

2.2. Vorgehen

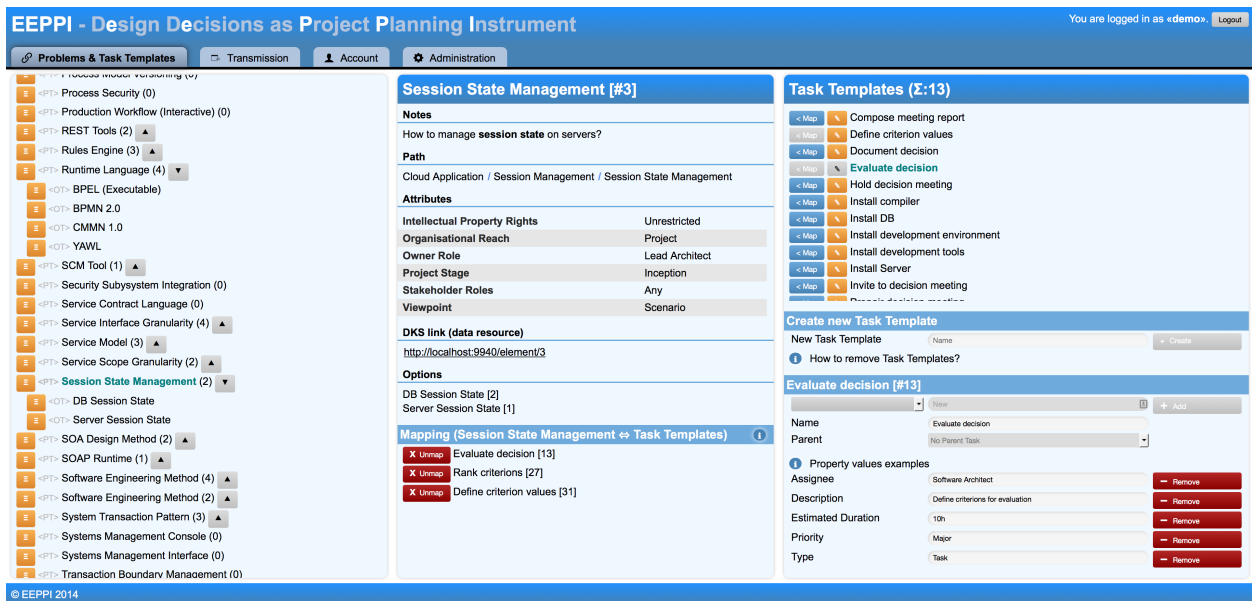
Aufbauend auf den Schnittstellen von Wissensverwaltungssystemen und Projektplanungstools wurden eine Applikation und eine Oberfläche entworfen, die eine flexible Konfiguration der Schnittstellen ermöglichen. Benutzer sollen Aufgabenvorlagen erstellen, diese mit Entscheidungen verknüpfen und in ein Projektplanungstool übertragen können.

Mittels Prototyp wurde die Machbarkeit dieses Ansatzes überprüft und anschliessend im Rahmen mehrerer Iterationen eine Webapplikation entwickelt. Zusammen mit dem Betreuer, als Ansprechpartner der Kundengruppe, wurden Usability- und Workflowtests durchgeführt, um Benutzeroberfläche und Datenfluss vom Entscheidungsverwaltungs-

system bis ins Projektplanungstool zu validieren. Abschliessend folgte zur Stabilisierung eine Überarbeitungsphase.

2.3. Ergebnis

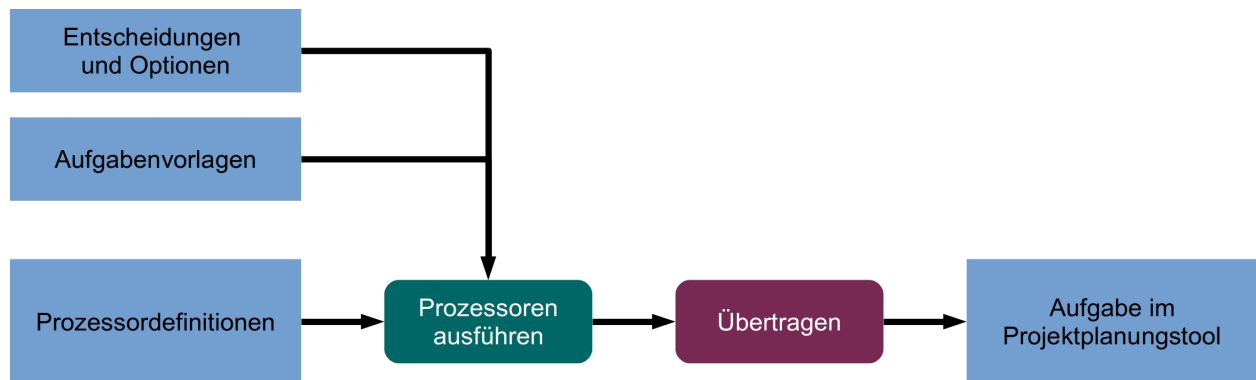
Im Rahmen der Arbeit wurde eine Webapplikation entwickelt, die mögliche Entscheidungen aus einem angebundnen Wissensverwaltungssystem bezieht und dem Benutzer mit einem Metamapping ermöglicht, Projektentscheidungen mit eigenen Aufgaben zu verknüpfen.



Metamapping in EEPPi: Verknüpfen von Entscheidungen und Aufgabenvorlagen

Die Applikation besteht aus einem Server- und einem Clientteil. Der Serverteil übernimmt die Speicherung der Daten und bietet eine Schnittstelle für den Client an. Der Clientteil ist für die Darstellung der Benutzeroberfläche und für die Steuerung der Applikation zuständig.

Über einen Administrationsbereich konfiguriert der Benutzer die Applikation nach seinen Bedürfnissen. Beispielsweise kann der Benutzer selbst den Aufbau der zu generierenden Aufgaben steuern. Ein dazu entwickelter Templatingmechanismus ermöglicht ihm, eigene Verarbeitungsfunktionen, sogenannten Prozessoren, zu verwenden.



Übertragen: Ausführen von Prozessoren und Übermitteln der Aufgaben an ein Projektplanungstool

EEPPI zeigt, was kommerzielle Produkte in diesem Bereich anbieten könnten, aber auch die Design-Herausforderungen einer solchen Software: Sowohl die Verknüpfung von Entscheidungen und Aufgaben wie auch die Anbindung an die umliegenden Systeme müssen sehr flexibel gestaltet sein. Mit EEPPI ist dies bereits gelungen; es gibt jedoch noch viele Erweiterungsmöglichkeiten. EEPPI legt somit einen wichtigen Meilenstein im Forschungsbereich des interdisziplinären Entscheidungs- und Projektmanagements und zeigt einen möglichen Entwicklungspfad für zukünftige Tools auf.

Inhaltsverzeichnis

1	Abstract	10
2	Management Summary	11
2.1	Ausgangslage	11
2.2	Vorgehen	11
2.3	Ergebnis	12
3	Problemstellung	18
3.1	Ziele	18
3.2	Abgrenzung	18
3.3	Optionale Erweiterungen	18
3.4	Mehrwert gegenüber bestehenden Lösungen	19
4	Projektorganisation	20
4.1	Projektteam	20
4.2	Projektbegleitung	20
4.3	Projekt- und Qualitätsmanagement	21
5	Lösungskonzept	22
5.1	Übersicht	22
5.2	Anbinden der externen Systeme	22
5.2.1	Anbinden von Wissensverwaltungssystemen	23
5.2.2	Anbinden von Projektplanungstools	25
5.3	Tier-Architektur	25
5.4	Technologie	27
5.4.1	Server	27
5.4.2	Client	32
5.4.3	Testing	35
5.5	Session Management	35
5.6	Datenfluss	36
5.7	Entwurf, Begründungen und Domain Model	37
5.7.1	Konzeptionelle Domäne	38
5.7.2	Metamapping	40
5.7.3	Implementationsdomäne	41
5.7.4	Umbenennen und Löschen von Domänenobjekten	43
5.7.5	Tasktemplates Strukturierung	43
5.7.6	Verknüpfungen von Tasktemplates und Entscheidungs-Vorlagen	46
5.7.7	Übertragung von Tasks in ein Projektplanungstool	46

5.7.8	Transmission-Workflow	47
5.7.9	Mapping Methode	51
5.7.10	Kommunikation	51
6	Schnittstellen und Protokolle	53
6.1	RESTfull HTTP Schnittstelle	53
6.2	Dokumentation des API	53
6.2.1	Herkunft der Daten	54
6.2.2	Beispielaufrufe	55
6.3	Client	56
6.4	HTTP Verben	56
7	Benutzeroberfläche	57
7.1	Userinterface-Mocking	57
7.2	Finales Userinterface	59
8	Lizenzen und verwendete externe Produkte	63
8.1	EEPPI	63
8.2	Verwendete Frameworks und Libraries	63
8.3	Verwendete Tools und Technologien	64
8.4	Lizenzen	64
9	Ergebnisse	66
9.1	Zielerreichung	66
9.1.1	Hauptziel	66
9.1.2	Rückblick auf die Fragestellungen der Aufgabenstellung	67
9.1.3	Erfolgsfaktoren	68
9.2	Umsetzung der Userstories	69
9.3	Offene Punkte, Bugs, technische Probleme	69
9.4	Zusätzliche Features	70
9.4.1	API-Dokumentation	70
9.4.2	Strukturierung von Tasktemplates	70
9.4.3	Im- und Export der Daten von EEPPI	70
9.4.4	Beispielrequest für Redmine	70
9.5	Erweiterungs-Möglichkeiten	71
9.5.1	Erweiterung der bestehenden Applikation	71
9.5.2	Ersatz der Serverapplikation	71
9.5.3	Ersatz der Clientapplikation	71
9.5.4	Erstellung eines Proxies	71
9.5.5	Verwendung der API	72
9.6	Mögliche Erweiterungen	72
9.6.1	Rechte und Rollen	72
9.6.2	Vererbende Tasktemplates	72
9.6.3	Reporting	72
9.6.4	Import/Export	73
9.6.5	Undo von übertragenen Tasks	73
9.6.6	Bearbeitungsmöglichkeiten für zu übertragende Tasks	73

9.6.7	Kaskadierende Processors	73
9.6.8	System Status Benachrichtigungen	73
9.6.9	Verwendung mehrerer Entscheidungswissenssysteme	73
9.6.10	Verwendung mehrerer Projekte	74
9.6.11	Unterscheidung von verschiedenen Projektplanungstools	74
9.6.12	Rückfluss von Informationen des Tasks zurück in Tasktemplate	74
9.6.13	Rückfluss von Informationen ins Entscheidungswissenssystem	74
9.6.14	Weitere Projektplanungstool-Schnittstellen	75
9.7	Zukunft von EEPPI	75
10	Glossar	76
A	Anforderungen	82
A.1	Allgemeine Beschreibung	82
A.1.1	Benutzer-Charakteristik	82
A.1.2	Einschränkungen	82
A.2	User Stories	83
A.2.1	Personas	83
A.2.2	Definitionen	83
A.2.3	Übersicht über die User Stories	85
A.3	Weitere Anforderungen	90
A.3.1	Qualitätsmerkmale	90
A.4	Sicherheit	91
B	Projekt- und Qualitätsmanagement	93
B.1	Projektmanagement	93
B.2	Qualitätssicherung	96
B.3	Testkonzept	96
B.4	Testergebnisse	98
B.5	Metriken	99
B.5.1	Umfang	99
B.5.2	Test Coverage	101
B.5.3	Qualität	101
B.5.4	Fazit	103
C	Infrastruktur	105
C.1	Arbeitsplätze, Geräte und Server	105
C.2	Tools	105
C.2.1	Projektmanagement	105
C.2.2	Versionsverwaltung	105
C.2.3	Literaturverwaltung	106
C.2.4	Dokumentation	106
C.2.5	Dokumentation des API	107
C.2.6	Code Dokumentation	108
C.2.7	Modelling	108
C.2.8	UI Mockups	108
C.2.9	Building & Testing	108

C.2.10 Individuelle Entwicklungsumgebungen	108
C.3 Backup	109
C.3.1 Persönliche Entwicklungsgeräte	109
C.3.2 Arbeitsplätze im Zimmer 1.206	109
C.3.3 Virtual Server	109
C.3.4 Repository und Entwicklungsserver	109
D EEPPI-Usermanual	110
D.1 Beispieldaten für Request-Templates	121
E Code-Documentation	124
E.1 API	124
E.2 Client Code	148

3. Problemstellung

«Lassen sich aus Projektentscheidungen Aufgaben ableiten?» Mit dieser Frage befasst sich die vorliegende Bachelorarbeit EEPPI.

Sowohl das Treffen einer Entscheidung wie die Entscheidung zugunsten einer Wahlmöglichkeit implizieren Aufgaben in einem Projekt. So führt die Entscheidung «Welche Art Session State soll verwendet werden?» zum Beispiel zu den Aufgaben «Session State evaluieren» und «Prototyp umsetzen». Wird bei dieser Entscheidung die Option «Database Session State» ausgewählt, so resultieren aus diesem Entscheid beispielsweise die Aufgaben «Datenbank installieren» und «Session Persistenz implementieren».

Aufgaben unterscheiden sich abhängig von Kontext, Unternehmung, Zeitplanung und Detaillierungsgrad der Entscheidung. Daraus schliessen wir, das eine Applikation, die das Erzeugen von Aufgaben aus Projektentscheidungen ermöglichen will, eine hohe Flexibilität und Konfigurierbarkeit aufweisen muss.

3.1. Ziele

Es soll herausgefunden werden, ob sich ein Metamapping formulieren lässt, das Entscheidungen und Aufgaben verknüpft. Weiter soll ermittelt werden, ob dieser Prozess automatisierbar und in einer Software umsetzbar ist. Dazu soll eine Webapplikation entwickelt werden, die Entscheidungen aus einem Entscheidungswissenssystem bezieht, dem Benutzer für das Metamapping ein Werkzeug anbietet und Aufgaben in ein Projektplanungstool exportiert.

3.2. Abgrenzung

Die bestehenden Systeme sollen dabei nicht verändert werden. So sollen keine Informationen aus dem Projektplanungstool zurück ins Entscheidungswissenssystem fließen. Auch die Zuständigkeitsbereiche der einzelnen Tools sollen klar getrennt bleiben. So soll EEPPI keine Funktionalität zum Modellieren von Entscheidungen enthalten und auch keine Projektplanungsfunktionalität integrieren. Zudem soll der Fokus der Applikation im Bereich der Konzepte und deren Umsetzung liegen. Die Implementierung von Nebenfunktionen wie zum Beispiel Rechte- und Rollenkonzepten besitzt entsprechend eine tiefere Priorität und soll daher schlank umgesetzt werden.

3.3. Optionale Erweiterungen

Optional sind beispielsweise eine Strukturierung von Aufgaben oder Funktionalität zum Import und Export von Aufgaben und Verknüpfungen denkbar. Ebenfalls denkbar ist die Anbindung mehrerer Entscheidungswissenssysteme und eine Verschmelzung deren Entscheidungen oder Mandantenfähigkeit der Applikation.

3.4. Mehrwert gegenüber bestehenden Lösungen

Lösungen zur Verknüpfung von Entscheidungsmanagement und Projektplanung sind ein aktuelles Forschungsthema, entsprechend gibt es noch keine uns bekannten Produkte auf dem Markt.

4. Projektorganisation

4.1. Projektteam

Für das Projekt verantwortlich sind Laurin Murer und Tobias Blaser. Sie sind beide Informatik-Studenten an der HSR und schliessen das Studium im Winter 2014/2015 ab.



Abbildung 4.1.: Laurin Murer



Abbildung 4.2.: Tobias Blaser

4.2. Projektbegleitung

Betreuer des Projekts ist Prof. Dr. Olaf Zimmermann. Er ist Dozent an der HSR mit Schwerpunkt Software Architecture, Enterprise SOA und Cloud [16]. Er betreut die Arbeit, begleitet das Team durch regelmässige Meetings und ist Ansprechpartner der Kundengruppe.



Abbildung 4.3.: Prof. Dr. Olaf Zimmermann

4.3. Projekt- und Qualitätsmanagement

Zur Verwaltung des Projektes setzt das Projektteam das Projektplanungstool Jira von Atlassian¹ ein.

The screenshot displays the Jira System Dashboard with the following sections:

- Filter Results: My Work in Progress**: No matching issues found.
- Filter Results: My Issues to Review**:

T	Key	P	Summary	Due
	BA-162	↑	Write personal feedbacks	12.12.14
	BA-85	↑	BA-8 / Write and send meeting report from 15.12.2014	
- Filter Results: My Further Assigned Issues**:

T	Key	P	Summary	Fix Versions	Due
	BA-227	↑	Rework whole document	Release BA	15.12.14
	BA-240	↑	Check references between decisions (ref)	Release BA	15.12.14
	BA-241	↑	Check HSR rules for forgotten things	Release BA	15.12.14
	BA-242	↑	Compose personal feedback of tobias	Release BA	15.12.14
	BA-108	↓	Inform user about network/dks/ppt connection		
- Filter Results: Unassigned Issues (this version)**:

T	Key	P	Summary	Due
	BA-232	↑	Copy projekt plan to documentation	15.12.14
	BA-229	↑	Check HSR rules for BA's for forgotten things and epics	15.12.14
	BA-228	↑	Check all screenshots for update	15.12.14
	BA-10	↑	Technical BA documentation	15.12.14
- Favourite Filters**:

Advanced user stories	6
Core user stories	7
Documentation	24
My Further Assigned Issues	5
My Issues to Review	2
My Work in Progress	0
Open Issues	241
Open or TODO	20
Projectmanagement	63
Ready to Review without Reviewer	0
Stories	13
Unassigned Issues (this version)	4
- Activity Stream**:
 - Your Company JIRA**
 - Tobias Blaser** closed BA-208 - Fix some wording issues (Yesterday, Comment, Watch)
 - Tobias Blaser** closed BA-233 - Document possible future life of EPPPI (Yesterday, Comment, Watch)
 - Tobias Blaser** closed BA-243 - Compose project review of laurin (Yesterday, Comment, Watch)
 - Tobias Blaser** logged '3 hours, 30 minutes' on BA-227 - Rework whole document (Review conclusion, licenses, future) (Yesterday, Comment, Watch)

Abbildung 4.4.: Atlassian Jira System Dashboard

Im Abschnitt B auf Seite 93 wird das Projekt- und Qualitätsmanagement mit Jira und Git detailliert erläutert.

¹<https://www.atlassian.com/software/jira>

5. Lösungskonzept

Nachfolgend werden die grundlegenden Konzepte von EEPPI, Architektur, gewählte Technologien und die EEPPI-Domäne erläutert.

5.1. Übersicht

EEPPI soll Entscheidungswissenssysteme (DKS¹) mit Projektplanungstools (PPT²) verbinden.

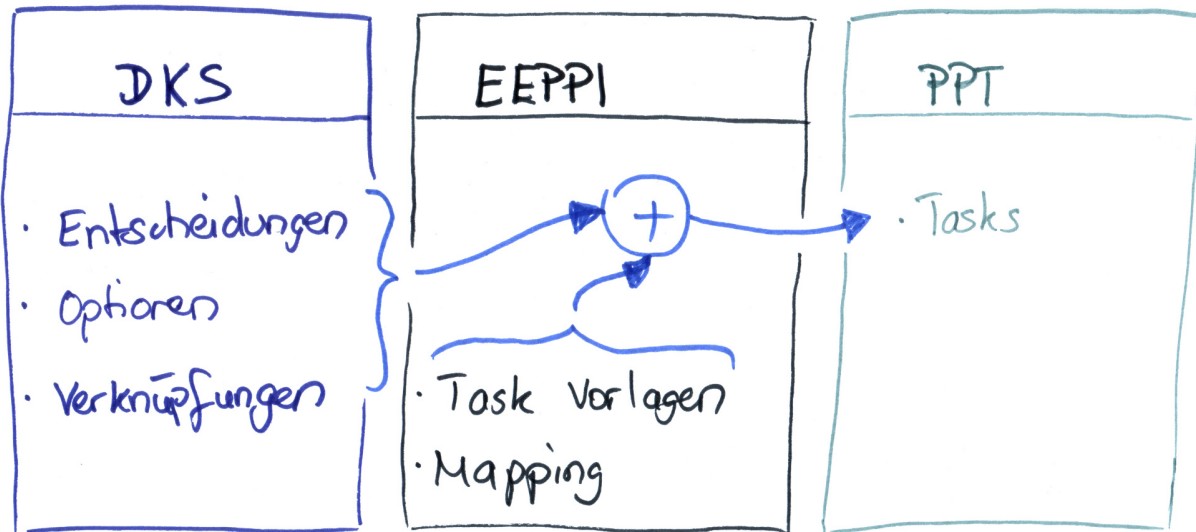


Abbildung 5.1.: Architektur Übersicht

Dabei sollen Entscheidungen sowie deren verknüpfte Wahlmöglichkeiten (Optionen) mit Task Vorlagen (Tasktemplates) verknüpft werden. Diese Verknüpfung wird in folgenden Kapiteln als «Mapping» bezeichnet. Anhand diesen Verknüpfungen sollen aus den Aufgabenvorlagen konkrete Aufgaben generiert und in ein Projektplanungstool übertragen werden.

5.2. Anbinden der externen Systeme

Damit EEPPI Daten aus Wissensverwaltungssystemen laden und in Projektplanungstools übertragen kann, muss EEPPI an diese angebunden werden können.

¹Decision Knowledge System, beispielsweise CDAR

²Project Planning Tool, beispielsweise Atlassian Jira

5.2.1. Anbinden von Wissensverwaltungssystemen

Beim CDAR handelt es sich um ein Wissensverwaltungssystem, das im Rahmen einer Bachelorarbeit [12] an der HSR entwickelt wurde. Mit dem CDAR kann ein Benutzer Entscheidungen und Optionen modellieren und verknüpfen. Das CDAR wurde in der Anfangsphase des Projektes als Entscheidungswissenssystem verwendet.

Für die Anbindung von EEPPI ans CDAR gibt es verschiedene Möglichkeiten. Wir haben uns für eine eigene Serverkomponente, ein eigenes Userinterface und eine eigene Persistenz entschieden. Die Alternativen sowie die Grundlagen dieser Entscheidung sind nachfolgend aufgeführt.

Bei den folgenden verwendeten Entscheidungstemplates handelt es sich um Templates gemäss dem «IBM UMF Template for Decision Log» [4].

Themengebiet	Architekturdesign	Thema	Integration
Name	Erweiterung CDAR / Integration	ID	INT-CDAR
Getroffene Entscheidung	Eigene Serverkomponente, eigenes UI (keine UI Integration mit CDAR), eigene Persistenz		
Problemstellung	In welcher Art soll EEPPI mit dem CDAR integriert werden?		
Voraussetzung	Die CDAR API Stellt alle benötigten Daten zur Verfügung.		
Motivation	Diese Entscheidung beeinflusst die Möglichkeiten der Technologiewahl der zu nutzenden Schnittstellen und Komponenten und ist daher Grundlegend für weitere Entscheidungen.		
Alternativen	<p>Jede der Entscheidungen (Serverkomponente, Clientapplikation, Persistenz) kann in diesem Fall unabhängig der andern zwei getroffen werden. Darum sind hier nur die Alternativen der jeweiligen einzelnen Entscheidungen und nicht alle Kombinationen aufgelistet:</p> <p>CDAR UI erweitern Integration der EEPPI-Funktionalität ins UI des CDAR. Vorteile Nur eine Applikation für Benutzer Nachteile CDAR UI muss angepasst werden, EEPPI ist vom CDAR abhängig</p> <p>Serverkomponente ersetzen EEPPI bildet eine gemeinsame neue Serverkomponente, die diejenige des CDAR ersetzt. Vorteile Einheitlichen Unterbau für CDAR und EEPPI, nur eine Schnittstelle, nur eine Serverkomponente, einfachere Installation Nachteile Sehr aufwändig, da die CDAR Server Komponente viel zu ersetzende Logik beinhaltet, EEPPI ist mit CDAR gekoppelt.</p> <p>CDAR Persistenz erweitern EEPPI nutzt die Persistenz des CDAR und erweitert diese. Vorteile Einfachere Wartung, nur eine Persistenz für Backup Nachteile Kopplung von EEPPI an CDAR</p>		
Begründung	<p>EEPPI soll die CDAR-API zum Laden der Daten benutzen, jedoch eine eigene Server- sowie UI-Komponente und eine eigene Persistenz besitzen.</p> <p>Vorteile</p> <ul style="list-style-type: none"> • Die Persistenz kann im gleichen System wie CDAR untergebracht sein, kann aber auch auf einem komplett andern Host laufen. • Es sind keine Anpassungen an CDAR notwendig, weder an der Persistenz, der Serverkomponente noch am UI. • EEPPI ist Unabhängig vom CDAR und könnte auch mit einer andern Applikation als das CDAR gekoppelt werden. <p>Nachteile Benutzer müssen zwei Applikationen nutzen (andere URL als CDAR), die Installation ist komplizierter</p>		
Annahmen	CDAR soll als Referenz-DKS angebunden werden.		
Abgeleitete Anforderungen	Die Schnittstelle für die Datenquelle (Anbindung CDAR) muss generisch und konfigurierbar gestaltet sein.		
Verknüpfte Entscheidungen	«Tier-Architektur» Abbildung 5.4 Seite 27, «Server Technologie» Abbildung 5.9 Seite 31		

Abbildung 5.2.: Erweiterung CDAR / Integration

Im Laufe der Prototypenphase wurde seitens des Betreuers als Vertreter der Kundengruppe entschieden, CDAR durch die schlanke Schnittstellenapplikation ADRepo zu ersetzen, die ihre Daten über die Enterprise Architect Erweiterung ADMentor bezieht. Diese Veränderung bestätigte die Entscheidung für die gewählte Variante. Das Team plante, die Authentisierungsschnittstelle des CDAR zu nutzen um Benutzern einen Single Sign On zu ermöglichen. Die Ablösung des CDAR bedingte allerdings, dass EEPPI selbst eine Benutzerverwaltung aufbauen muss.

5.2.2. Anbinden von Projektplanungstools

Entscheidend für die Anbindung eines Projektplanungstool ist deren Schnittstelle zum Erzeugen von Tasks. Als Referenz-Projektplanungstools haben wir uns für Jira und Redmine entschieden.

Beide Systeme besitzen offene und gut dokumentierte Schnittstellen, die funktional sehr mächtig sind und im Fall von Redmine sogar mehrere Formate (JSON und XML) unterstützen. Dadurch ermöglichen beide Systeme das Verwalten von Tasks über die Schnittstelle. Weiter besitzen sie einen hohen Verbreitungsgrad, auch über die Softwarebranche hinaus.

5.3. Tier-Architektur

Die Tier-Architektur wurde zusammen mit der Wahl der Technologie durchgeführt aufgrund der gegenseitigen Einflüsse und Einschränkungen. Technologische Entscheidungen werden im Abschnitt 5.4 auf Seite 27 besprochen.

Aufgrund der Technologischen Möglichkeiten und den anzubindenden Schnittstellen stehen drei mögliche Tier-Architekturen zur Auswahl. Die hier verwendeten Begriffe stützen sich auf von Prof. Dr. Zimmermann verwendete Begriffe in der HSR Vorlesung «Application Architecture» [10].

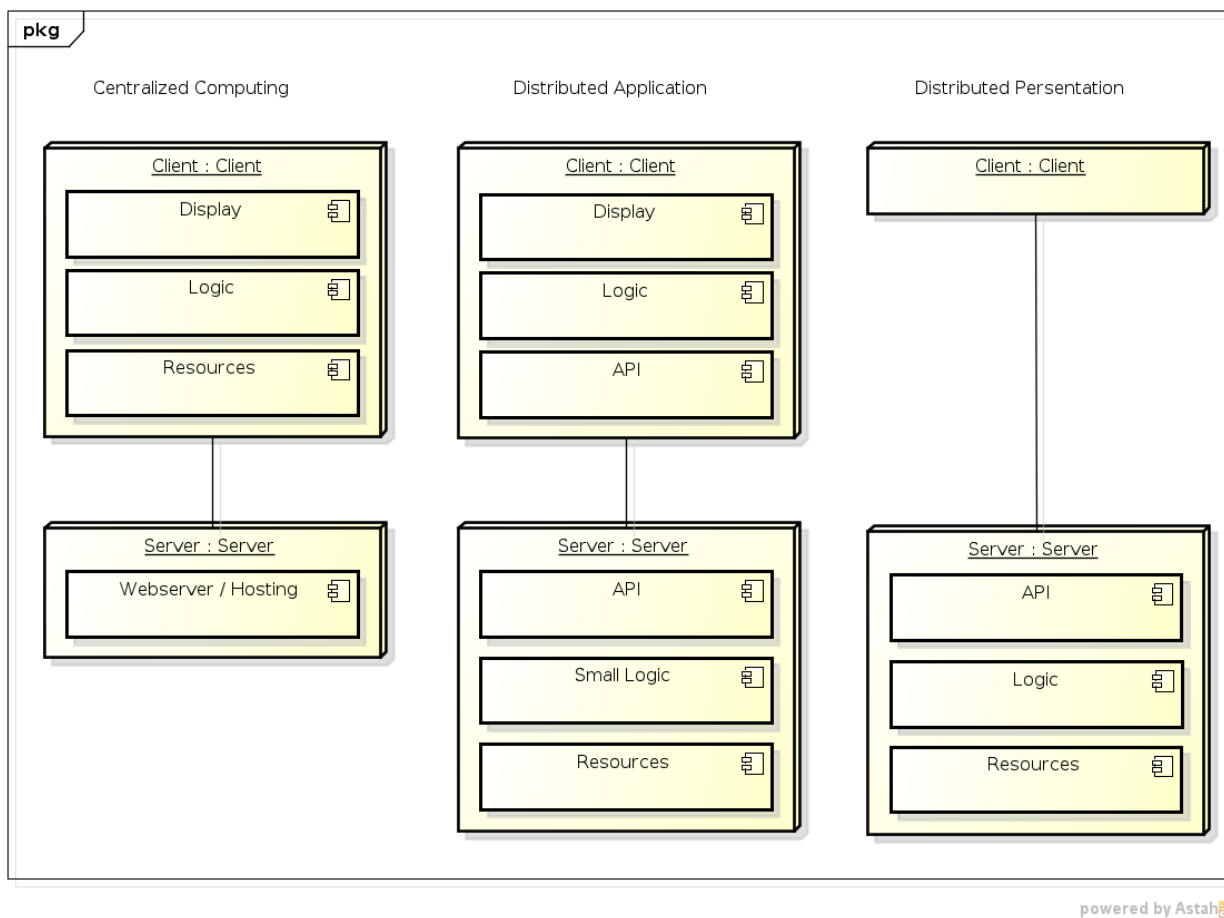


Abbildung 5.3.: Architektur Varianten

- 1-Tier Structure: Centralized Computing (Client-only Application)** Die Serverkomponente übernimmt lediglich das Ausliefern einer WebApp. Die WebApp bezieht die Daten direkt aus externen Schnittstellen. Persistenz findet dezentral auf dem Client statt in Form von File Persistence oder Persistence durch das Framework (zum Beispiel HTML5 Storage).
- 2-Tier Structure: Distributed Application (Single Page App)** Die Serverkomponente übernimmt Persistenz sowie minimale Logik (zum Beispiel Login). Präsentation und Logik werden von der Client Komponente übernommen.
- 2-Tier Structure: Distributed Presentation** Persistenz, Logik und Präsentation werden vom Server übernommen. Die Präsentation wird fertig aufbereitet an den Client gesendet (zum Beispiel HTML Page). Es gibt keine aktiven Komponenten auf dem Client, ausgenommen asynchron nachladende Skripte.

Themengebiet	Architekturdesign	Thema	Architektur
Name	Tier-Architektur	ID	ARC-TIERS
Getroffene Entscheidung	Distributed Application (Single Page Application)		
Problemstellung	Welche Tier-Architektur soll für EEPPI gewählt werden?		
Voraussetzung	Die einzusetzende Technologie unterstützt «Centralized Computing», «Distributed Presentation» und «Distributed Application»		
Motivation	Diese Entscheidung ist wichtig, damit eine möglichst lose Kopplung & hohe Flexibilität auch für zukünftige, auf EEPPI aufbauende, Applikationen erreicht wird und der Grundstein für Technologieentscheidungen gelegt wird.		
Alternativen	«Centralized Computing», «Distributed Presentation»		
Begründung	«Distributed Application» ermöglicht eine Serverseitige (zentralisierte) Persistenz sowie den Aufbau einer schnellen und unabhängigen Applikation durch Nutzung der Rechenleistung auf dem Client, sodass der Server und dessen Kosten schlank gehalten werden können. Da die App vom Server ausgeliefert wird, kann sie zentral von dort aus verwaltet, gewartet und kontrolliert werden.		
Annahmen	keine Speziellen		
Abgeleitete Anforderungen	Die Serverkomponente muss Persistenz sowie eine Datenschnittstelle für den Client bereitstellen.		
Verknüpfte Entscheidungen	«Server Technologie» Abbildung 5.9 Seite 31, «Client Framework» Abbildung 5.11 Seite 34		

Abbildung 5.4.: Tier-Architektur

5.4. Technologie

Für die Umsetzung von EEPPI haben wir uns für verschiedene Technologien entschieden. Die Evaluation sowie die Technologien selbst werden nachfolgend beschrieben.

5.4.1. Server

Die Erstellung einer Webanwendung ist Teil der Anforderungen von EEPPI. Als Erstes wird im folgenden Abschnitt die Auswahl der Servertechnologie beschrieben.

5.4.1.1. Evaluation

Die Servertechnologie stellt eine entscheidende Komponente dar, ihre Wahl ist von entsprechender Tragweite für andere Komponenten des Projektes. Aus diesem Grund haben wir sie ausführlich evaluiert.

1. Schritt: Gemeinsame Definition von Kriterien und möglichen Servertechnologien.

2. Schritt: Separate Festlegung der Gewichtung der Kriterien sowie Definition der Matrix Kriterium/Servertechnologie mit der persönliche Schätzung, wie stark jede Servertechnologie das entsprechende Kriterium erfüllt.
3. Schritt: Zusammenführung der persönlichen Ergebnisse zu Gesamtergebnis

Kriterium	Priorität		Normierte Priorität		Definitive Priorität	
	Tobias	Laurin	Tobias	Laurin		normiert
Gleiche Technologie wie bestehende Applikation	1	1	5.4%	2.6%	4	4.5%
Gleiche Technologie Server/Client	1	1	5.4%	2.6%	4	4.5%
Kenntnisse im Team	0.5	5	2.7%	12.8%	8	9.1%
Typisierte Sprache	2	5	10.8%	12.8%	12	13.6%
Präferenz des Dozenten	1	5	5.4%	12.8%	8	9.1%
Offene Lizenz	1	5	5.4%	12.8%	6	6.8%
Benötigt wenig Boiler Plate Code	1	2	5.4%	5.1%	6	6.8%
Macht Spass zum Entwickeln	2	5	10.8%	12.8%	12	13.6%
Unterstützung von Hosting Providern	4	2	21.6%	5.1%	8	9.1%
IDE-Unterstützung Mac/Linux	1	4	5.4%	10.3%	8	9.1%
Zukunftsaussichten des Frameworks	4	4	21.6%	10.3%	12	13.6%
Total	18.5	39	100%	100%	88	100%

Abbildung 5.5.: Servertechnologie-Vergleich: Prioritätsfindung

Abbildung 5.5 zeigt den Prozess der Prioritätsfindung. Wie bereits erwähnt, haben wir zuerst die Kriterien definiert und anschliessend je separat Prioritäten für jedes Kriterium festgelegt. Da wir verschiedene Skalen verwendet haben, haben wir unsere Punkte noch zu 100% normiert. Und aus diesen zwei Werten haben wir in gemeinsamer Diskussion die definitive Priorität erstellt. Dies hat meist problemlos funktioniert, ausser bei den Punkten «Kenntnisse im Team» und «Unterstützung von Hosting Providern» (blau markiert) hatten wir zu Beginn nennenswerte Unterschiede. Bei diesen Punkten haben wir dann in der gemeinsamen Diskussion einen Wert festgelegt.

Kriterium	Priorität	CDAR erweitern	eigener Tomcat	Play Framework	Node.js JavaScript	Node.js TypeScript	Ruby on Rails	PHP, Flow, Doctrine
Gleiche Technologie wie bestehende Applikation	4.5%	1	1	0.5	0	0	0	0
Gleiche Technologie Server/Client	4.5%	0	0	0	1	1	0	0
Kenntnisse im Team	9.1%	0.5	0.5	0.8	0.75	0.6	0.1	0.75
Typisierte Sprache	13.6%	1	1	1	0	1	0	0.25
Präferenz des Dozenten	9.1%	1	1	1	0	0	0	0
Lizenz		Apache 2	Apache 2	Apache 2	MIT	MIT	MIT	LGPL v3+, MIT
Offene Lizenz	6.8%	1	1	1	1	1	1	0.5
Benötigt wenig Boiler Plate Code	6.8%	0.25	0.4	0.8	0.8	0.8	0.9	0.8
Macht Spass zum Entwickeln	13.6%	0.25	0.5	1	0.25	1	1	0.5
Unterstützung von Hosting Providern	9.1%	0.5	0.5	0.5	0.25	0.25	0.25	1
IDE-Unterstützung Mac/Linux	9.1%	1	1	1	1	1	1	1
Zukunftsaussichten des Frameworks	13.6%	0.5	0.5	1	1	1	1	1
Total		61%	69%	85%	52%	75%	53%	58%

Abbildung 5.6.: Servertechnologie-Vergleich: Vergleich der Technologien

Abbildung 5.6 zeigt den Vergleich der einzeln evaluierten Servertechnologien. Auch hier haben wir zuerst je separat die Schätzung gemacht und dann verglichen. Bei diesem Vergleich haben wir sehr ähnliche Werte gewählt, lediglich bei den Zukunftsaussichten von PHP, Flow und Doctrine (blau markiert) hatten wir nennenswert unterschiedliche Ansichten.

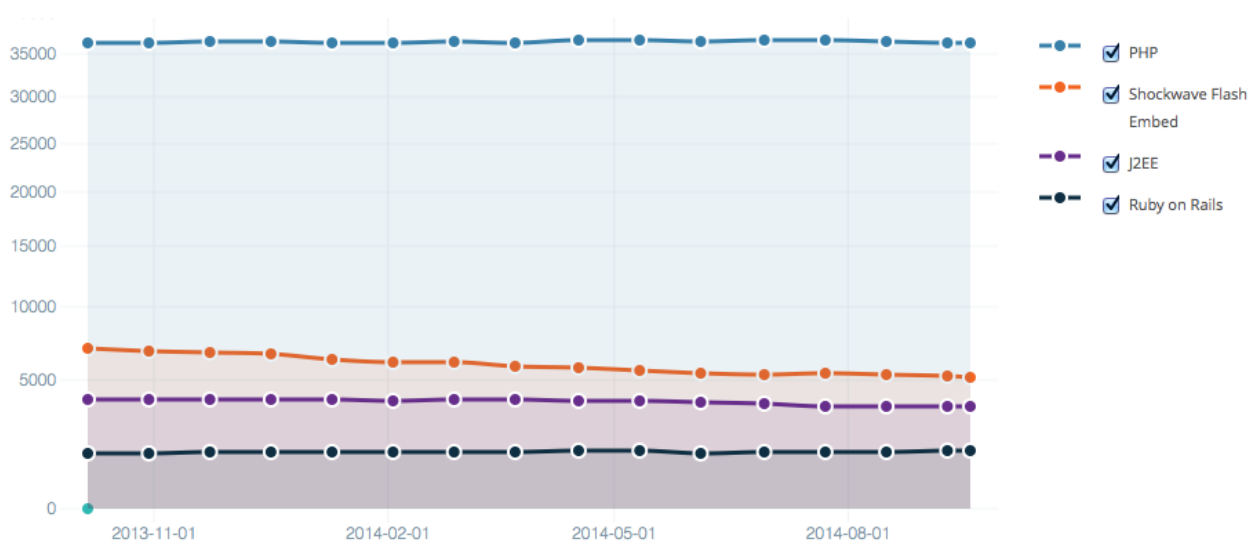


Abbildung 5.7.: Entwicklung von Webserver-Technologien der Top 10'000 Sites [6]

Schlussendlich haben wir uns nach Recherchen auf eine «1» geeinigt, weil entgegen den Erwartungen von Laurin Murer sich die Verbreitung von PHP (auch bei grösseren Seiten) kaum verändert hat in den letzten Jahren. Als Beispiel für eine Technologie, die immer weniger eingesetzt wird und in den nächsten Jahren verschwinden wird, haben wir Shockwave Flash herangezogen, welche, wie in Abbildung 5.7 sichtbar, im letzten Jahr deutlich an Boden verloren hat. Im Vergleich dazu ist PHP sehr gut im Markt vertreten und besitzt auch eine äusserst konstante Verbreitung. Deshalb, und unter dem Gesichtspunkt der soliden Konzepte, die den in den letzten Jahren entstandenen modernen PHP Frameworks wie Doctrine und Flow zugrunde liegen, sehen wir da auch das Fortbestehen für diese Technologien gegeben.

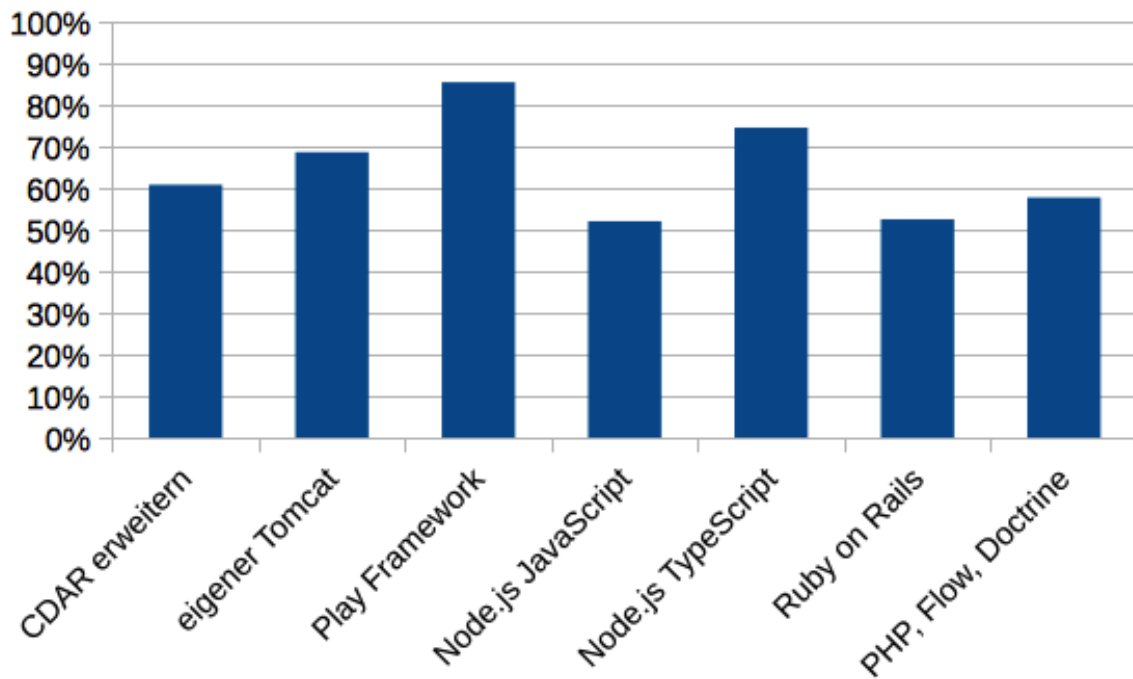


Abbildung 5.8.: Ergebnis Serverttechnologie-Vergleich

5.4.1.2. Ergebnis

Schlussendlich haben wir für jede Technologie das Total der Punkte berechnet (Punkte jedes Kriteriums multipliziert mit seiner Priorität). In Abbildung 5.8 ist das Ergebnis abgebildet. Aufgrund diesem haben wir uns für das Play Framework entschieden. Eine Alternative wäre noch Node.js mit TypeScript gewesen, diese erreichte jedoch, aufgrund der Präferenz des Betreuers für Java und der schlechteren Unterstützung durch Hostingangebote, 10% weniger Punkte.

Themengebiet	Server	Thema	Technologie
Name	Servertechnologie	ID	TEC-SERVER
Getroffene Entscheidung	Play Framework		
Problemstellung	Welche Servertechnologie soll eingesetzt werden?		
Voraussetzung	Die einzusetzende Technologie erlaubt die ausgewählte Tier-Architektur-Variante		
Motivation	Von der Servertechnologie hängen unter Anderem die Entscheidungen für Server und Persistenz Framework ab.		
Alternativen	CDAR erweitern, eigener Tomcat, Node.js+JS, Node.js+TS, Ruby, PHP		
Begründung	<ul style="list-style-type: none"> • Das Play Framework basiert auch auf Java, wie das bestehende CDAR. • Ein Teammitglied hat bereits Erfahrungen mit dem Framework gemacht. • Das andere Teammitglied kennt die zugrundeliegende Sprache (Java). • Der Betreuer hat eine Präferenz für Java. • Das Play Framework benötigt wenig Boilerplate-Code^a. • Es sprechen keine zwingenden Gründe dagegen (wie beispielsweise zu stark einschränkende Lizenzen). • Das Play Framework ist angenehm zum Entwickeln für die Entwickler: gegebener Spass-Faktor, Vermeidung von Fehlern, Unterstützung im Erreichen von guter Code Qualität. • Im IFS^b ist Know How zum Play Framework vorhanden (M. Stocker). <p>^aCode, der (wiederholt) geschrieben werden muss, ohne dass er wirklich etwas zum Projekt beiträgt</p> <p>^bInstitut für Software, HSR Hochschule für Technik: http://www.ifs.hsr.ch/</p>		
Annahmen	keine		
Abgeleitete Anforderungen	Potentielle Hosting Provider müssen Java unterstützen		
Verknüpfte Entscheidungen	«Tier Architektur» Abbildung 5.4 Seite 27		

Abbildung 5.9.: Servertechnologie

5.4.1.3. Play Framework

Kapitel 5.4.1.1 erläutert den Hintergrund der Entscheidung für das Play Framework. Wikipedia beschreibt das Play Framework wie folgt:

«Play is an open source web application framework, written in Scala and Java, which follows the model–view–controller (MVC) architectural pattern. It aims to optimize developer productivity by using convention over configuration, hot code reloading and display of errors in the browser.» [6]

5.4.2. Client

Nachfolgend wird die Auswahl der Technologien für die Clientseite der Applikation beschrieben.

5.4.2.1. Sprache

JavaScript ist die einzige clientseitige Sprache, die von allen Browsern ohne Installation eines Plugins unterstützt wird. Entsprechend haben wir in diesem Bereich keine Wahlmöglichkeit. Es gibt jedoch einige Precompiler³, die verschiedene Vor- und Nachteile bieten.

³Compiler, der Code einer andern Sprache in JavaScript umwandelt, bevor dieser ausgeführt wird.

Themengebiet	Sprache	Thema	Technologie
Name	Clientseitige Programmiersprache	ID	TEC-CLIENT-LANG
Getroffene Entscheidung	TypeScript		
Problemstellung	Welche Sprache soll clientseitig verwendet werden?		
Voraussetzung	Der eingesetzte Build-Server unterstützt Precompiling für JavaScript oder bietet das starten eigener Skripte an		
Motivation	Diese Entscheidung beeinflusst die Optionen der Client Frameworks und sie entscheidet, ob die gewählte Sprache gutes Code-Schreiben unterstützt, wenig fehleranfällig ist und Fehler frühzeitig erkannt werden können, oder ob die Sprache Fehler begünstigt und damit die Qualität des Codes verringert.		
Alternativen	<p>JavaScript</p> <p>Vorteile UI der bestehenden Applikation ist auch in JavaScript geschrieben</p> <p>Nachteile Fehler tauchen erst zur Runtime auf, sprachspezifische Besonderheiten, die Fehler begünstigen</p> <p>Dart</p> <p>Vorteile Moderne, optional typisierte, kompilierte Programmiersprache (wird zu JS kompiliert)</p> <p>Nachteile Benötigt Dart VM oder Dart-to-JS Transcompiler, wenig verbreitet</p>		
Begründung	<p>TypeScript wird zu JavaScript kompiliert, sodass die Vorteile beide Sprachen kombiniert werden. TypeScript versucht, möglichst den Standard zukünftiger ECMA-Script Versionen einzuhalten, sodass die Kompatibilität immer gewährleistet und die Syntax möglichst ähnlich bleiben soll.</p> <p>Vorteile Optisch besser lesbar als JavaScript, verhindert bekannte, häufige Probleme in JavaScript wie zum Beispiel das ungewohnte Verhalten von «this», verbessert die Codequalität gegenüber JavaScript massiv, da viele Fehler zur Kompilierzeit gefunden werden</p> <p>Nachteile Erfordert TSC^a-Compiler, Code Overhead bei Inheritance</p> <p>^aTypeScript Compiler, kann über den Node-Packagemanager von Node.js installiert werden</p>		
Annahmen	keine		
Abgeleitete Anforderungen	Die Entwicklungsumgebung benötigt Node.js für den TypeScript Compiler.		
Verknüpfte Entscheidungen	«Tier Architektur» Abbildung 5.11 Seite 34		

Abbildung 5.10.: Clientseitige Programmiersprache

5.4.2.2. Architektur-Framework

Um clientseitig eine grössere Applikation aufzubauen, gibt es verschiedene Möglichkeiten. Einerseits kann selbst ein Mini-Framework entwickelt werden, andererseits gibt es viele weit entwickelte Frameworks mit grossem Funktionsumfang.

Themengebiet	Framework	Thema	Technologie
Name	Clientseitiges Applikationsframework	ID	TEC-CLIENT-FW
Getroffene Entscheidung	AngularJS		
Problemstellung	Welches Applikationsframework soll clientseitig eingesetzt werden?		
Voraussetzung	Es wird eine clientzentrierte oder verteilte Applikation entwickelt		
Motivation	Diese Entscheidung legt den Grundstein für den technologischen Aufbau der clientseitigen Applikation und soll ein Framework evaluieren, das Templating und Two-Way-Binding mit UI Komponenten unterstützen, um dies nicht selbst implementieren zu müssen.		
Alternativen	<p>EmberJS</p> <p>Vorteile Sehr modular und anpassbar, weniger Overhead als AngularJS</p> <p>Nachteile Bringt wesentlich weniger mit als Angular JS, mehr Eigenaufwand notwendig, kann nur einfache Datentypen an UI Elemente binden</p> <p>Kein Framework</p> <p>Vorteile Vollständig freie Architekturgestaltung</p> <p>Nachteile Hoher Implementationsaufwand ohne Gewinn</p>		
Begründung	<p>AngularJS ist ein bekanntes MVW^a- und Templating-Framework, erlaubt eine saubere Trennung von Logik und Darstellung und bindet ViewModel-Eigenschaften und -Funktionen ans Template. Dadurch lassen sich Observer-Konstrukte sparen. AngularJS ist stabil, zuverlässig, gut erweiterbar und bringt von sich aus schon sehr viel mit gegenüber EmberJS. Es wurde auch schon für das bestehende Entscheidungswissenssystem CDAREingesetzt.</p> <p>^aModel View Whatever: Fasst MVC, MVP und andere ähnliche Strukturierungspatterns zusammen, wird manchmal auch «MV*» genannt.</p>		
Annahmen	keine		
Abgeleitete Anforderungen	Die Serverkomponente muss eine Schnittstelle bieten, über welche die Clientapplikation Daten austauschen kann.		
Verknüpfte Entscheidungen	«Client Sprache» Abbildung 5.10 Seite 33, «Tier Architektur» Abbildung 5.4 Seite 27		

Abbildung 5.11.: Clientseitiges Applikationsframework

5.4.2.3. JavaScript Autoloading

Ursprünglich wurde diskutiert, Require.js als Autoloader zu benutzen. Require.js bietet nebst Vorteilen wie Namespacing und Autoloading leider den grossen Nachteil, dass jedes Skript einzeln asynchron vom Client angefordert wird, was insbesondere bei grösseren Round-Trip-Times die Applikation spürbar verlangsamt. Aus diesem Grund wurde entschieden, auf die Module von TypeScript zur Strukturierung zu setzen und die verschiedenen Skripte bereits beim Kompilieren zusammenzuführen.

5.4.2.4. Styling

Das Frontend der Clientapplikation wird mit CSS⁴ gestaltet. Um dieses einfacher schreiben zu können, wird Less⁵ als CSS Preprocessor⁶ eingesetzt, da es schlankeren Sourcecode ermöglicht und Vorteile wie Variablen und Mixins bietet. Less wird serverseitig kompiliert um den Client zu entlasten.

5.4.3. Testing

Das Testframework für den Clientteil soll einfach einzubinden und zu erweitern sein. Mit «erweitern» ist in diesem Falle gemeint, neue Tests und Testsuits hinzuzufügen. Ebenfalls soll es eine Syntax ähnlich der Assert-Syntax von JUnit bieten und mit einem Build Server gekoppelt werden können.

5.4.3.1. Jasmine

Jasmine arbeitet mit einem realen Browser (keine Browsersimulation), ist einfach zu handhaben und bietet typische Assert-Syntax. Ausserdem wird es direkt von AngularJS promoted.

5.5. Session Management

Eine Session (Session State Pattern⁷) kann auf dem Client, auf dem Server oder in einer Datenbank abgelegt werden.

⁴Cascading Style Sheets: Auszeichnungssprache für Webdokumente, um diese grafisch zu gestalten

⁵CSS Pre-Prozessor: <http://lesscss.org/>

⁶Vorverarbeitungsschritt, bei dem Code einer andern Sprache in CSS umgewandelt wird

⁷<http://www.bettersoftwaredesign.org/Design-Patterns/Enterprise-Application-Architecture-Patterns/Session-State-Patterns>

Themengebiet	Architekturdesign	Thema	Architektur
Name	Session State	ID	ARC-SESSION
Getroffene Entscheidung	Client Session State		
Problemstellung	Auf welchem Tier sollen User Sessions gespeichert werden?		
Voraussetzung	Sowohl das Server- wie das Clientframework beherrschen Session State Management		
Motivation	Diese Entscheidung beeinflusst Client wie Server und bestimmt, ob der der Server zustandsbehaftet oder nicht ausgelegt wird.		
Alternativen	«Server Session State», «Database Session State»		
Begründung	«Client Session State» ermöglicht die Umsetzung eines zustandslosen Servers. Dadurch wird eine reine Ressourcen-basierte Serverschnittstelle möglich.		
Annahmen	Es wird eine Identifizierung/Wiedererkennung des Benutzers benötigt		
Abgeleitete Anforderungen	Der Client muss eine Möglichkeit bieten, eine Session zu speichern. Bevorzugt soll dafür ein Session Cookie zum Einsatz kommen.		
Verknüpfte Entscheidungen	«Server Technologie» Abbildung 5.9 Seite 31, «Client Framework» Abbildung 5.11 Seite 34		

Abbildung 5.12.: Session State

5.6. Datenfluss

Die Übertragung der Daten von und zu den externen Systeme läuft jeweils über den EEPPI-Server, welcher dafür die Rolle eines CORS⁸ Proxy einnimmt. Dies ist nötig, da Clients nicht ohne zusätzliche Erlaubnis eines Remoteservers auf diesen zugreifen dürfen (Cross Origin Restriction). Dies ist ein allgemeines Problem und tritt bei vielen Webanwendungen auf.

Anstatt einen eigenen CORS Proxy zu verwenden, hätte auch ein Drittanbieterservice verwendet werden können. Als Beispiels sei hier corsproxy.com genannt. Dies ist ein CORS Proxy in Form eines Webservices. Ein CORS Proxy akzeptiert Remote Requests von irgendwelchen Ursprungsadressen und leitet diese dann an die entsprechenden Server weiter, die keine Cross-Origin-Requests erlauben.

Wir haben uns entschieden selbst einen kleinen CORS Proxy in unsern Server zu integrieren, damit die Daten nicht über fremde Services fließen und EEPPI auch in einem lokalen Netzwerk mit beschränktem Internetzugriff betrieben werden kann. Da EEPPI Daten mit Businessrelevanz verarbeitet, ist die Vertraulichkeit der Daten eine wichtige Komponente, was ebenfalls gegen einen externen Service spricht.

⁸Cross-Origin Resource Sharing: http://de.wikipedia.org/wiki/Cross-Origin_Resource_Sharing

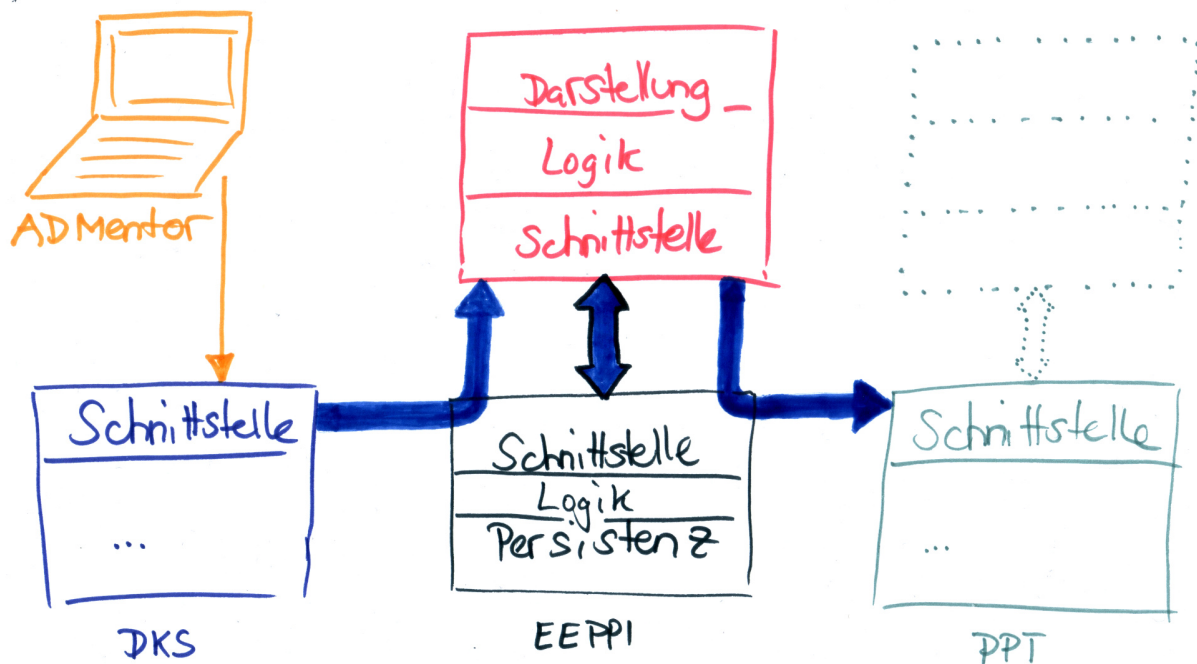


Abbildung 5.13.: Applikationsdatenfluss mit Beispieldatenquelle ADMentor

5.7. Entwurf, Begründungen und Domain Model

Nachfolgend werden die wichtigsten Bezeichnungen von Domänenobjekten eingeführt:

Problem Beschreibt eine Vorlage für ein Designproblem eines Software Projektes, zum Beispiel «Session State». Im Entscheidungswissenssystem ADRepo werden Problems als «Problem Template» bezeichnet.

Alternative Beschreibt eine Vorlage für eine Wahlmöglichkeit eines Problems. Für das Problem «Session State» wären dies zum Beispiel «Server Session State» und «Database Session State». Im Entscheidungswissenssystem ADRepo werden Alternatives als «Option Template» bezeichnet.

Decision Beschreibt eine konkrete Problem-Instanz. Decisions entstehen, wenn Problems auf konkrete Projekte angewendet werden. Decisions sind sowohl geschlossene wie noch zu treffende Entscheidungen. Im Entscheidungswissenssystem ADRepo werden Decisions als «Problem Occurrences» bezeichnet.

Option Beschreibt eine Wahlmöglichkeit einer Decisions und somit eine konkrete Instanz einer Alternative. Eine Option kann gewählt, nicht gewählt oder noch offen sein. Im Entscheidungswissenssystem ADRepo werden Options als «Option Occurrences» bezeichnet.

Tasktemplate Beschreibt eine Vorlage zum Erstellen von konkreten Tasks (Aufgabenvorlage). Tasktemplates enthalten generische Werte, wie zum Beispiel «Project Manager» als Attributwert für die Eigenschaft «Assignee».

Task Beschreibt einen aus einem Tasktemplate erzeugten konkreten Task. Task entstehen während dem Übertragen der Informationen eines Tasktemplates an ein Projektplanungstool.

Mapping Bezeichnet die Verknüpfung von Problems oder Alternatives mit einem Tasktemplate. Anhand dieser Verknüpfung werden die, für die Übertragung eines Tasks an ein Projektplanungstool, benötigten Daten erstellt.

Requesttemplate Bezeichnet eine Vorlage für einen HTTP-Request um in einem spezifischen Projektplanungstool Tasks anzulegen. Requesttemplates beinhalten Platzhalter (Variablen und Funktionen, sog. Processors), die mit Daten der Tasktemplates und Decisions oder Options ersetzt werden.

Processor Verarbeitungsfunktion (zu dt. Prozessor), die Daten eines Mappings verarbeitet und einen Rückgabewert liefert, der anstelle der Processorsignatur ins Requesttemplate eingefügt wird.

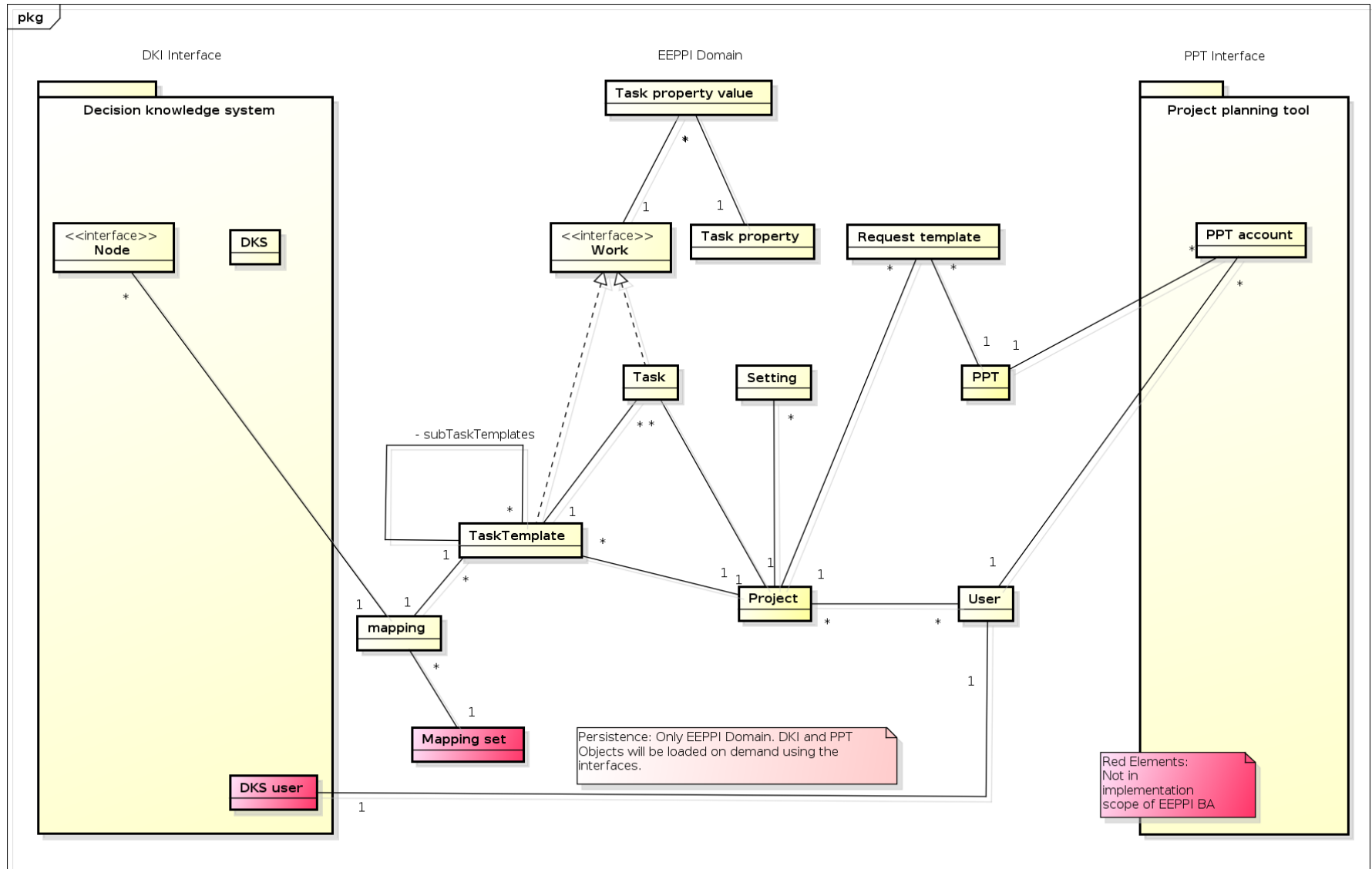
5.7.1. Konzeptionelle Domäne

Die EEPPI Domäne setzt sich aus drei Typen von Objekten zusammen:

- Einer Abstraktion der Objekte hinter der Schnittstelle der angebundnen Entscheidungswissensverwaltung
- Einer Abstraktion⁹ der Objekte hinter der Schnittstelle eines oder mehreren angebundnen Projektplanungstools
- Den eigenen Objekten und Schnittstellen

Innerhalb von EEPPI werden nur die eigenen Objekte persistiert. Die Objekte der andern System werden On-Demand über die Schnittstellen geladen.

⁹In Form des, durch das Request-Template gebundenen, Schnittstellenaufrufs



powered by Astah

Abbildung 5.14.: Konzeptionelle EEPPI-Domäne

EEPPI arbeitet mit den Objekten des Entscheidungswissenssystem. Das ADRepo als Referenz-Entscheidungswissenssystem ist wie folgt aufgebaut:

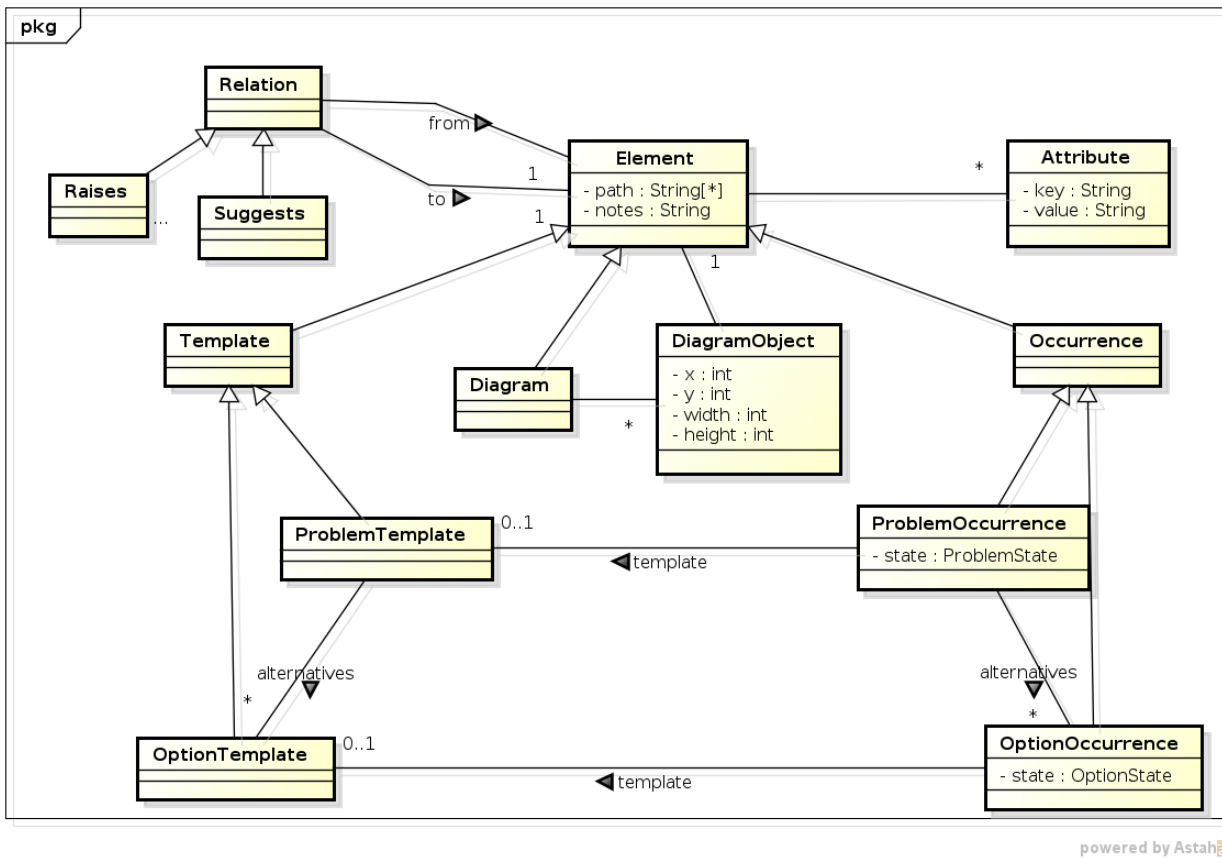


Abbildung 5.15.: ADRepo-Domäne (Quelle: IFS HSR, siehe Abbildungsverzeichnis)

Dabei werden die Objekte von EEPPI wie folgt gemappt:

Element Node

ProblemTemplate Problem

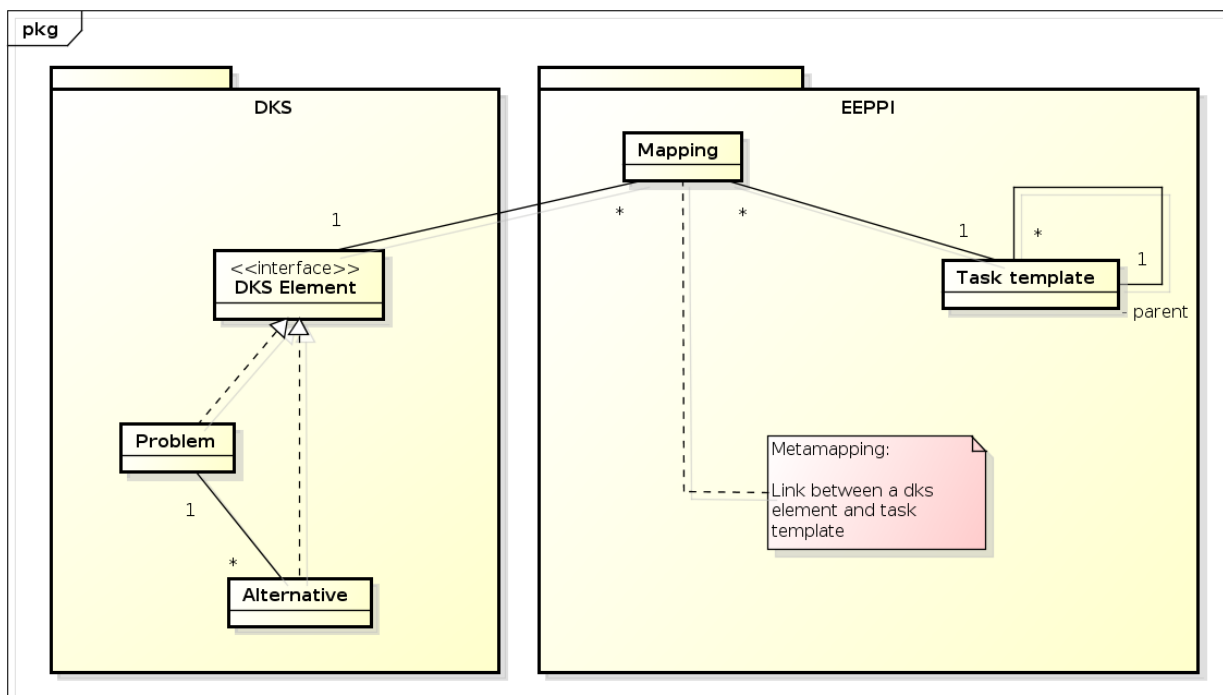
ProblemOccurrence Decision

OptionTemplate Alternative

OptionOccurrence Option

5.7.2. Metamapping

Metamapping bezeichnet das Konzept, wie die Domäne des Entscheidungswissenssystems (DKS) und die EEPPI-Domäne verknüpft werden sollen. Übergreifend gesehen geht es dabei um nichts Geringeres als die Verbindung von Entscheidungsmanagement und Projektmanagement.



powered by Astah

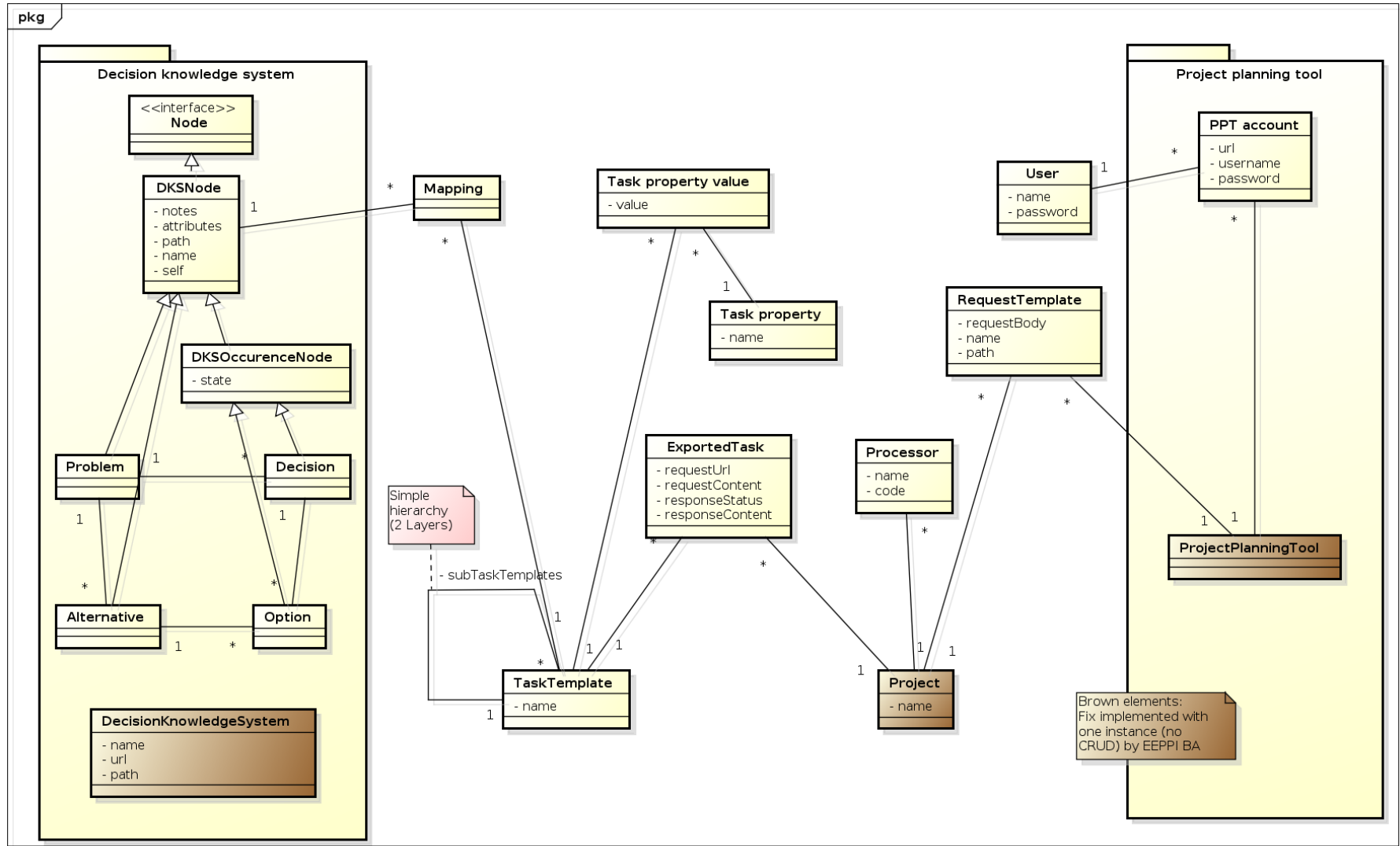
Abbildung 5.16.: Metamapping

Dazu wird die Entscheidungswissenssystem-Domäne von EEPPI abstrahiert und mit Tasktemplates verknüpft. Jedes Mapping steht für eine Verknüpfung eines Tasktemplates mit einem Problem oder einer Alternative.

Beim Erzeugen von konkreten Tasks müssen zu jedem Mapping alle Occurrences (siehe ADRepo-Domäne Abbildung 5.15) gefunden werden. Anschliessend können zusammen mit den Daten des Tasktemplates und dessen Eigenschaften (siehe EEPPI-Domäne Abschnitt 5.14) Tasks erzeugt werden.

5.7.3. Implementationsdomäne

Die konzeptionelle Domäne von EEPPI berücksichtigt auch mögliche Erweiterungsaspekte. Die Implementationsdomäne bricht aus der konzeptionellen die Teilmenge heraus, die effektiv umgesetzt wird und konkretisiert diese.



powered by Astah

Abbildung 5.17.: EEPPi-Implementationsdomäne

Ebenfalls enthält die Implementationsdomäne die Abstraktion der DKS Objekte. Sie zeigt auch das gegenüber der konzeptionellen Domäne einfacher gehaltene Benutzer- und Projekt-Konzept.

5.7.4. Umbenennen und Löschen von Domänenobjekten

Um Probleme mit referenzierten Domänenobjekten zu vermeiden, darf es Benutzern nur möglich sein, Tasktemplates umzubenennen, nicht jedoch zu löschen.

Das Löschen von referenzierten Tasktemplates würde dazu führen, dass ein Teil der Export-Historie verloren ginge und Mappings kein zugeordnetes Tasktemplate mehr besitzen würden. Dies wiederum würde zu fehlerhaften Exports, bzw. leeren Exports führen. Aus diesem Grund ist das Löschen von referenzierten Tasktemplates keine Option, höchstens das Löschen von noch nicht referenzierten.

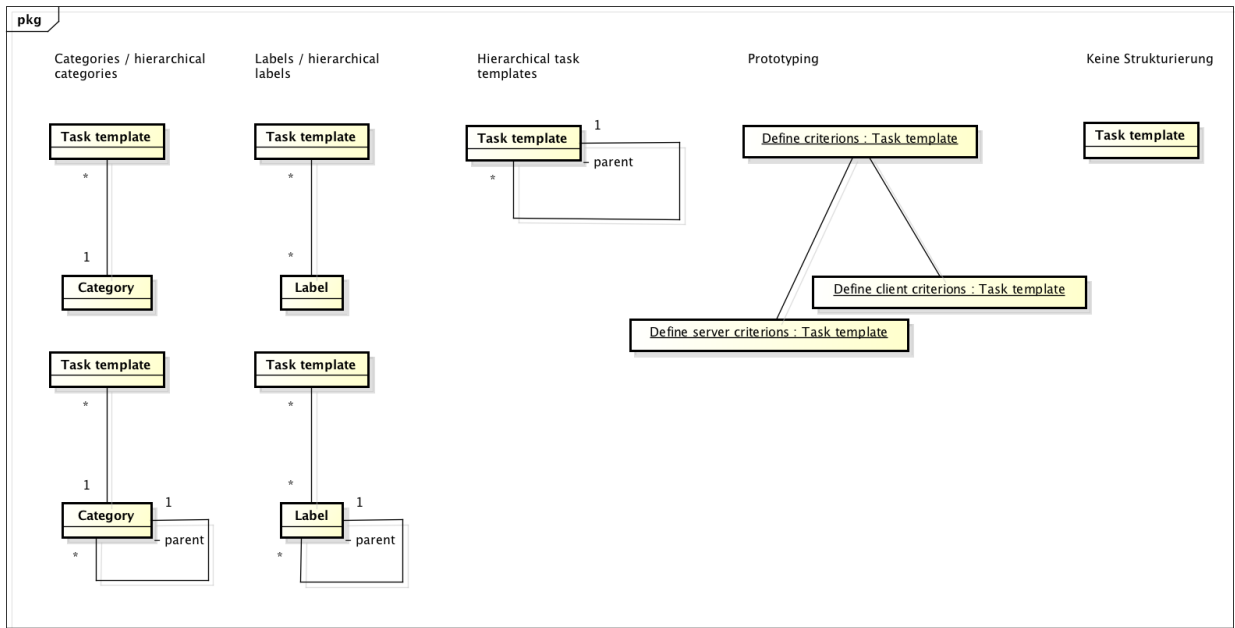
Könnten Benutzer Tasktemplates löschen, so müsste geprüft werden, ob diese referenziert werden. Sobald der Benutzer jedes Tasktemplate einmal exportiert hätte, könnte er ebenfalls keines mehr löschen. Die Möglichkeit zum Löschen ist entsprechend sowieso nur in einem neu aufgesetzten System gegeben.

Das konsequente Umsetzen der «Umbenennen statt Löschen»-Strategie ermöglicht Benutzern, falsch angelegte Tasktemplates weiterzuverwenden, vermeidet jedoch, dass sich die Applikation unterschiedlich verhält für bereits referenziert und noch nicht referenzierte Tasktemplates.

Das Gleiche gilt auch für Task Properties. Dies dürfen auf keinen Fall vom Benutzer entfernt werden und dürfen entsprechend nur umbenannt werden.

5.7.5. Tasktemplates Strukturierung

Es gibt verschiedene Möglichkeiten, Benutzer eine Strukturierung von Tasktemplates anzubieten. Tasktemplates können selbst in eine Struktur gebracht werden oder durch externe Strukturen geordnet werden.



powered by Astah

Abbildung 5.18.: Strukturierungsmöglichkeiten von Tasks

Themengebiet	Domain	Thema	Architecture
Name	Tasktemplates Strukturierung	ID	DOM-TT-STRUC
Getroffene Entscheidung	Keine Strukturierung, allenfalls auf Wunsch des Vertreters der Anforderungsgruppe nicht hierarchische Kategorien, Smart Filters		
Problemstellung	Welches Konzept soll zur Strukturierung von Tasktemplates eingesetzt werden?		
Voraussetzung	Projektplanungstool s unterstützen überhaupt hierarchische Tasks, ansonsten sind entsprechende Strukturierungsmöglichkeiten wenig sinnvoll		
Motivation	Benutzer sollen einfach und schnell erstellte Tasktemplates wieder finden.		
Alternativen	<p>Keine Strukturierung</p> <p>Vorteile Einfach zu implementieren, einfach verständlich für den Benutzer</p> <p>Nachteile Bei vielen Tasktemplates unübersichtlich, führt zu doppelten Tasktemplates, da existierende nicht gefunden werden.</p> <p>Labels/Hierarchische Labels Versehen der Elemente mit einem oder mehreren Labels. Benutzer können nach Labels suchen oder Filtern, um Vorlagen anzuzeigen.</p> <p>Vorteile Einfach verständlich für den Benutzer</p> <p>Nachteile Benutzer könnten zu faul sein, Labels anzulegen und zuzuordnen da es aufwändiger ist, als Kategorisieren</p> <p>Direkte Hierarchisierung der Tasktemplates Elemente werden direkt mit Eltern-elementen verknüpft und bilden einen hierarchischen Baum.</p> <p>Vorteile Einfach zu implementieren</p> <p>Nachteile Schwer verständlich für den Benutzer, da die Hierarchisierung unter Umständen nicht mit dem Workflow zusammenpasst</p> <p>Prototyping statt Strukturierung Elemente erben Funktionalität von einander, statt strukturiert zu werden.</p> <p>Vorteile Verringert die Anzahl Tasktemplates massiv, da Eigenschaften vererbt werden können</p> <p>Nachteile Schwieriger umzusetzen, schwieriger zu verstehen für Benutzer</p>		
Begründung	Keine Strukturierung hat für die Entwicklung sowie für den Benutzer Vorteile. So gibt es zu diesem Punkt kaum Entwicklungsaufwand und der Benutzer findet trotzdem dank Suchfunktionen die gesuchten Tasktemplates. Ausserdem muss er keine Zeit aufwenden um die Tasktemplates zu strukturieren. Falls Vertreter der Anforderungsgruppe jedoch eine weitergehende Strukturierung wünschen, empfehlen wir nicht-hierarchische Kategorien sowie Smart Filters. Kategorien erlauben eine Strukturierung auf einfache Weise. Smart Filter sind eine Ergänzung zu den beiden Optionen und ermöglichen das schnelle Finden anhand von Eigenschaften, ohne dass diese der Benutzer erfassen muss.		
Annahmen	keine		
Abgeleitete Anforderungen	Mögliche zukünftige Erweiterung durch Strukturierungsmöglichkeiten muss bei der Implementation berücksichtigt werden		
Verknüpfte Entscheidungen	keine		

Abbildung 5.19.: Tasktemplates Strukturierung

5.7.6. Verknüpfungen von Tasktemplates und Entscheidungs-Vorlagen

Wissensproduzenten können Tasktemplates mit Problems (Entscheidungs-Vorlagen) verknüpfen. Dabei kann und soll ein Tasktemplate verschiedenen Problems zugeordnet werden können. Ebenso können Problems natürlich mehrere Tasktemplates zugeordnet erhalten.

5.7.6.1. Arten der Zuordnung

Tasktemplates können mit Problems auf zwei Arten verknüpft werden:

1. Sie können dann fällig werden, wenn eine Entscheidung getroffen wurde (operativer Task).
2. Ein Tasktemplate dient dazu, Entscheidungen zu treffen (Entscheidungstask).

Auf die Tasktemplates selbst hat dies keinen Einfluss, sie sind unabhängig davon. Ob es sich um einen operativen Task oder einen Entscheidungstask handelt, hängt nur davon ab, ob das Tasktemplate mit einem Problem oder einer Alternative einer Entscheidung verknüpft ist.

Sollte eine Unterscheidung dennoch einmal notwendig sein, so kann dies mittels Processors umgesetzt werden (Siehe Abschnitt 5.7.8), da die Daten des Entscheidungswissenssystems Typeninformationen enthalten.

5.7.7. Übertragung von Tasks in ein Projektplanungstool

Aus Tasktemplates werden beim Übertrag in ein Projektplanungstool Tasks generiert.

Tasktemplates sind generische Vorlagen, die ständig weiterentwickelt werden sollen durch den Projektplaner. Aus diesem Grund wäre es unpraktisch, wenn ein Benutzer zur Änderung einer Vorlage für jedes Mapping alle Tasktemplates anpassen müsste. Die Anzahl Tasktemplates, die der Benutzer aktualisieren müsste, würde mit der Anzahl Mappings wachsen und schnell eine unüberblickbare Menge erreichen.

Darum werden Tasktemplates mit Mappings verknüpft (referenziert) und nicht kopiert zum Zeitpunkt des Mappings. Eine Kopie würde für jedes Mapping eine neue Instanz des Tasktemplates erzeugen, hätte jedoch zum Vorteil, dass für bereits exportierte Tasks weiterhin die vor der Änderung verwendete Version des Tasktemplates erhalten bliebe.

Passt der Benutzer nun ein Tasktemplate an und exportiert anschliessend Tasks, so wird für alle gemappten Problems das aktuelle Tasktemplate verwendet, weil deren Verknüpfungen nur auf ein einziges Tasktemplate zeigen.

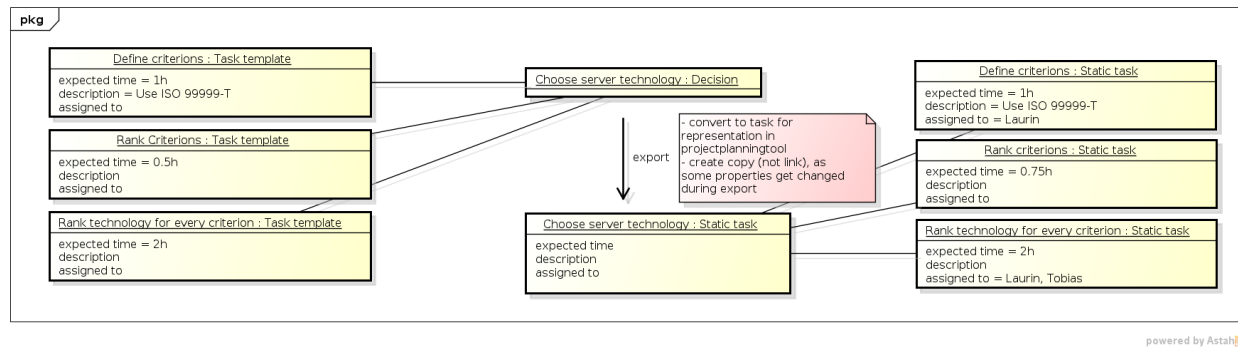


Abbildung 5.20.: Übertragen von Entscheidungen und Tasktemplates

Beim Übertragen werden aus Tasktemplates (konkrete) Tasks. Mit Alternatives verknüpfte Tasks werden entsprechend zu Sub-Tasks.

Benutzer wollen bei der Übertragung ins Projektplanungstool die vom Tasktemplate vorgegebenen Werte möglicherweise anpassen, wie zum Beispiel den erwarteten Aufwand für den Task. Daher ist es sinnvoll, die Eigenschaften der Tasktemplates in die (konkreten) Tasks zu kopieren, anstatt sie lediglich zu verknüpfen. Gleiches gilt für Eigenschaften von Entscheidungen. Würde jemand im Entscheidungswissenssystem diese verändern oder löschen, so würde dies die History zerstören.

5.7.8. Transmission-Workflow

Aus Tasktemplates erzeugte Tasks müssen zur Übertragung in ein Projektplanungstool umgewandelt werden, sodass die angesprochene API die Daten auch versteht. Dazu werden «Processors» eingesetzt.

5.7.8.1. Processors

Processors stellen kleine Funktionalitäten dar, die Daten umwandeln. Beispiele für Processors sind:

- «Date processor», der Kalenderdaten umwandelt
- «Issue type processor», der Issuetypen konvertiert
- «User processor», der Relationen zu Benutzern so umwandelt, dass das Projektplanungstool den User korrekt verknüpfen kann
- «Conditional processor» und «Option processor», die Bedingungen verarbeiten

Ebenfalls denkbar ist ein Processor, der Felder aggregieren kann und damit zum Beispiel nicht gemappte Felder in die Beschreibung überführen kann.

```
1 function(issueType) {
2   if(issueType == 'Feature') {
3     return 2;
4   } else if(issueType == 'Bug') {
5     return 3;
6   } else {
7     return 1; // 'Task'
8   }
9 }
```

Abbildung 5.21.: Beispielprocessor für Issuetypes

```
1 {
2   "issue": {
3     //...
4     {
5       //...
6       "issue_type": $issueTypeProcessor:(taskTemplate.attributes.Type)$,
7       //...
8     }
9   //...
10 }
```

Abbildung 5.22.: Processorverwendung in einem Requesttemplate

Das Codebeispiel zeigt einen Processor, der textuelle Issuetypes auf ID's des Projektplanungstool abbildet. Der im Template verwendete «issueTypeProcessor»-Aufruf wird mit dem Rückgabewert der Processorfunktion ersetzt, sodass für den Wert «Feature» für das Tasktemplate-Attribut «Type» die folgende Zeile resultiert:

```
1 "issue_type": 2,
```

Abbildung 5.23.: Requesttemplate nach dem Ausführen des Processors

Weitere Beispiele für Processors und eine Anleitung um selbst welche in ein Requesttemplate zu integrieren, finden sich im Abschnitt D, eine Auflistung über die zur Verfügung stehenden Inputdaten für Processors im Abschnitt D.1 im Anhang.

5.7.8.2. Variables

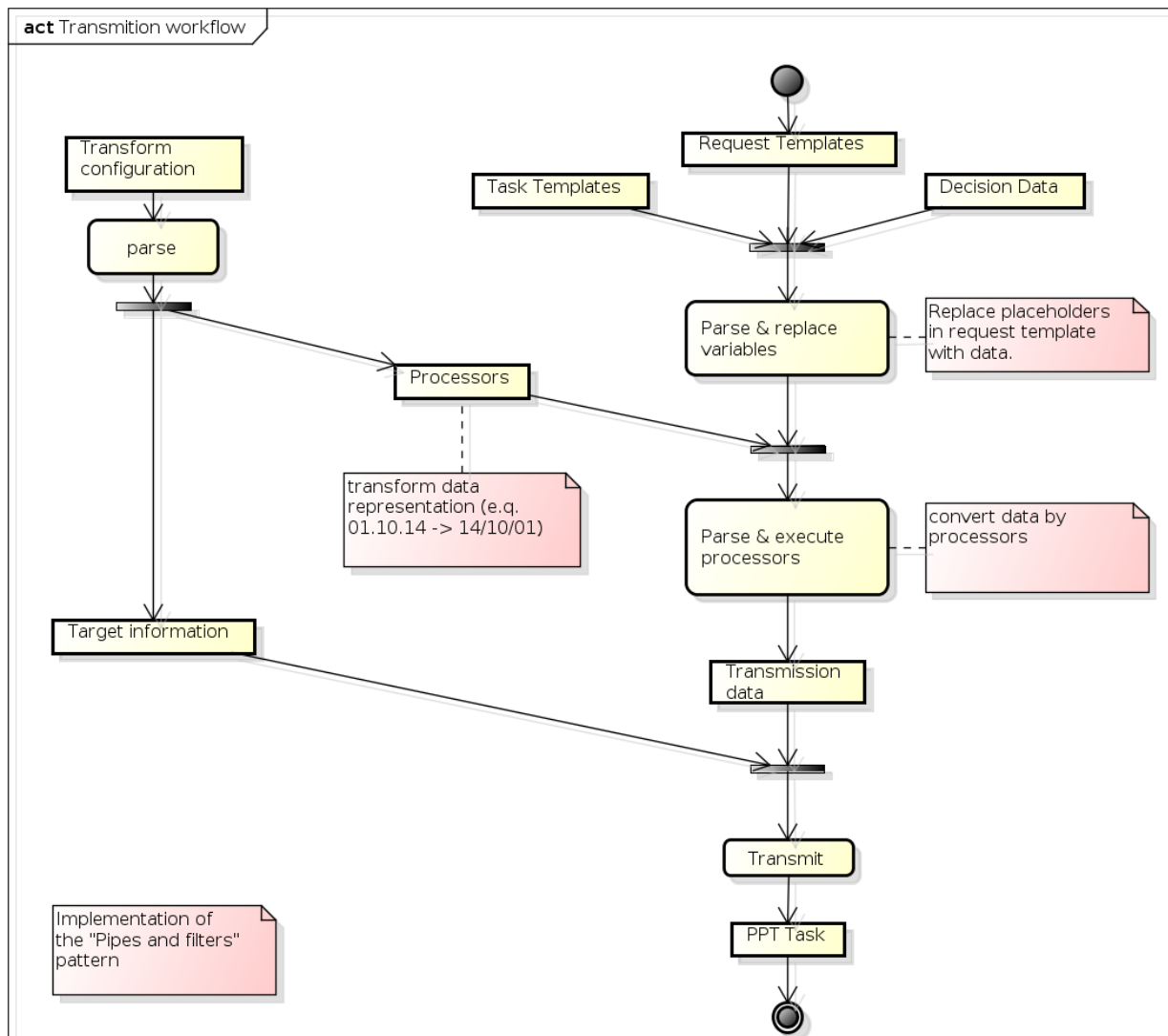
Variables können auf Daten von Tasktemplate und Decision zugreifen, diese aber nicht verändern.

```
1 //...
2 "estimated_time": ${taskTemplate.attributes.Duration},
3 //...
```

Abbildung 5.24.: Verwendung von Variables in einem Requesttemplate

5.7.8.3. Verarbeitung und Übertragung

Die folgende Grafik zeigt den kompletten Ablauf beim Verarbeiten und Übertragen von Tasks.



powered by Astah

Abbildung 5.25.: Übertragen von Tasks

Dem Transmissionworkflow liegt das «Pipes and filters»-Pattern zugrunde [5], welches eine Verarbeitungs- und Filterkette definiert. Die Processors stellen dabei die «Filters» dar.

Der komplette Transmittion-Workflow soll auf dem Client durchgeführt werden und nicht auf dem Server. Die Wahl der Servertechnologie schränkt die Möglichkeiten für dynamische Processors ein, während die Client Technologie dies ermöglicht.

Im Laufe der Erarbeitung dieses Workflows wurde über die Verarbeitung nicht gemappeter Eigenschaften diskutiert. In einer frühen Projektphase wurde entschieden, diese in Listenform in die Beschreibung des Tasks überzuführen. Die Entscheidung für ein sehr

flexibles Mapping führte dazu, dass dieser Anwendungsfall überflüssig wurde. Der Administrator kann selbst einen Processor definieren, der alle Felder in eine Liste überführt und zur Beschreibung des Task hinzufügt.

Die Vorteile dieser Lösung liegen auf der Hand: Die Entscheidung darüber, wie mit nicht gemappten Feldern umzugehen ist, kann der Administrator treffen und die Entwicklung unsererseits vereinfacht sich.

5.7.8.4. Secondary Processors und Variables

Processors und Variables werden vor dem Übertragen der Information ausgeführt. Informationen, die erst zur Zeit der Übertragung bekannt sind, können somit nicht durch Processors und Variables abgerufen werden.

Dafür sind sogenannte «Secondary Processors» notwendig, die direkt vor dem Übertragen ausgeführt werden. Diese können Informationen über hierarchisch höher stehende Tasks (Elterntasks) abrufen und für das Template aufbereiten. So ist beispielsweise eine Verknüpfung zwischen Eltern- und Sub-Tasks realisierbar.

Secondary Processors und Variables benutzen eine ähnliche Syntax¹⁰ wie gewöhnliche Processors und Variables, können jedoch nur auf die Daten der letzten Übertragung eines übergeordneten Tasks zugreifen.

```
1 {
2   "issue": {
3     //...
4     {
5       //...
6       "parent_issue_id": ${!parentRequestData.issue.id}$,
7       //...
8     }
9   //...
10 }
```

Abbildung 5.26.: Beispiel für die Verwendung einer Secondary Variable zum Verknüpfen des Eltern-Tasks

¹⁰Anstelle des \$ am Anfang beginnen sie mit \$!

5.7.9. Mapping Methode

Themengebiet	Domain	Thema	Architecture
Name	Mapping Methode	ID	DOM-TT-MM
Getroffene Entscheidung	Konfiguration/Block in Form von Templates mit Platzhaltern		
Problemstellung	Wie sollen Mappingkonfigurationen erstellt werden?		
Voraussetzung	Definiertes Konzept des Metamappings (Siehe Abschnitt 5.7.2)		
Motivation	Von der Mapping Method hängt die Architektur des Mappings und die Schnittstellen der Processors und Filters ab.		
Alternativen	<p>Hierarchische/Element basierte Konfiguration Das Mapping wird durch das Anlegen von verknüpften Elementen erzeugt.</p> <p>Vorteile Gegebene Validierung durch die Struktur, kein Parser notwendig</p> <p>Nachteile Aufwändiger umzusetzen, insbesondere das UI, weniger flexibel</p>		
Begründung	Eine Textblock/Template-basierte Konfiguration erhöht zwar die Fehlermöglichkeiten für den Administrator, ermöglicht diesem jedoch grössere Flexibilität und damit ein Abdecken einer grösseren Bandbreite an Projektplanungstools.		
Annahmen	Der Administrator kann mit Templates und Platzhaltern umgehen oder es lernen		
Abgeleitete Anforderungen	Das Mapping benötigt kein eigenes Datenmodell in Form von verknüpften Objekten. Es kann als einfache Text-Elemente an ein Projekt angeknüpft werden.		
Verknüpfte Entscheidungen	<p>Art der Speicherung in der Datenbank^a</p> <p>^aIn die Bachelorarbeit wurden nur die wichtigsten und grossen Entscheidungen eingefügt.</p>		

Abbildung 5.27.: Mapping Methode

Der Ablauf für einen Administrator sieht entsprechend wie folgt aus:

1. Projektplanungstool definieren
2. Taskeigenschaften erstellen
3. Mapping Taskeigenschaften -> Projektplanungstool erstellen

5.7.10. Kommunikation

EEPPI verwendet verschiedene Schnittstellen. Deren Verwendungen sind nachfolgend beschrieben.

5.7.10.1. Kommunikation zwischen EEPPI und dem Entscheidungswissenssystem

Die Übertragung zwischen dem Entscheidungswissenssystem und EEPPI basiert auf einer Ein-Weg-Kommunikation. EEPPI lädt die Daten, gibt jedoch keine Informationen

zurück, ob dies erfolgreich war oder nicht.

EEPPI schreibt auch keine Daten zurück ins Entscheidungswissenssystem, wie beispielsweise Informationen über die Verwendung der Problems und Alternatives. Grund dafür ist die dazu notwendige Komplexität der Schnittstelle sowie die mangelnde Unterstützung seitens Entscheidungswissenssystem. Auch würde diese Funktionalität den Rahmen der Arbeit sprengen, als mögliche Erweiterung von EEPPI ist sie jedoch denkbar.

5.7.10.2. Kommunikation zwischen EEPPI und dem Projektplanungstool

Die Übertragung der erzeugten Tasks ins Projektplanungstool basiert auf einer Zwei-Weg-Kommunikation. Zum einen werden die Tasks ans Projektplanungstool übertragen und zum anderen die Rückmeldung über die erfolgreiche Erzeugung der Tasks im Projektplanungstool wieder im EEPPI gespeichert. Ebenfalls zurückgeliefert werden minimale Informationen über den erstellten Task, wie zum Beispiel die ID. EEPPI verwendet diese Informationen, um dem Benutzer Secondary-Processors anzubieten, die zum Beispiel Informationen über einen Parent Task in das Requesttemplate einweben.

Der «Transmission»-Workflow als Gesamtes ist jedoch ein One-Way-Procedure. Dies bedeutet, dass keine Daten aus dem Projektplanungstool zurück in EEPPI fließen. Technisch zwar möglich, stellt eine Rückkopplung der Daten oder sogar eine Synchronisierung der Tasks eine grosse Herausforderung dar, die den Rahmen der Arbeit sprengen würde. Als mögliche zukünftige Erweiterung ist dies jedoch denkbar und könnte eines der Schlüsselkriterien werden, warum Projektplaner EEPPI einsetzen wollen.

6. Schnittstellen und Protokolle

6.1. RESTfull HTTP Schnittstelle

EEPPI besitzt eine RESTfull Schnittstelle, die andere Applikationen benutzen können, um EEPPI direkt anzusprechen. Diese Schnittstelle, auch API¹ genannt, wird auch von der eigenen Clientapplikation benutzt.

Das API unterstützt REST²-Level 2³. Das heisst, einzelne Ressourcen haben eigene Adressen und können unter Verwendung der HTTP-Verben GET, POST und DELETE benutzt werden.

6.2. Dokumentation des API

Sowohl für die Entwicklung des Clients wie auch für die weitere Entwicklung von EEPPI wurde das API des Servers ausführlich dokumentiert (siehe Abschnitt E.1). Ein Ausschnitt der API-Dokumentation ist in Abbildung 6.1 abgebildet. Das Konzept und die Methodik der Dokumentation wurde als Teil von EEPPI erstellt und ist strukturell an das API von Jira⁴ angelehnt.

¹Cascading Style Sheets

²Representational State Transfer

³REST Maturity Model: http://de.wikipedia.org/wiki/Representational_State_Transfer#REST_Maturity_Model

⁴<https://docs.atlassian.com/jira/REST/latest/>

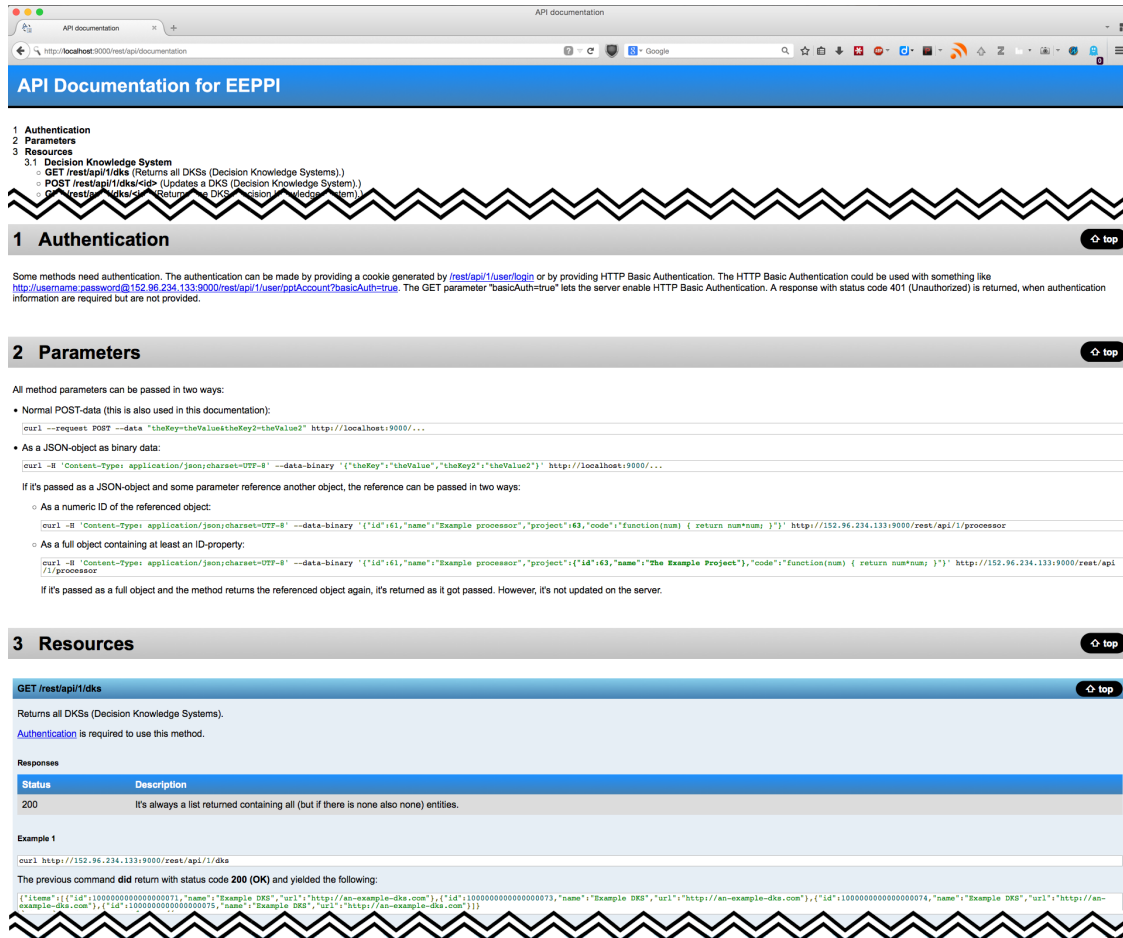


Abbildung 6.1.: API-Dokumentation im Browser (Vollständige Dokumentation siehe Abschnitt E.1)

6.2.1. Herkunft der Daten

Damit das API mit möglichst geringem Aufwand jederzeit den neusten Entwicklungsstand repräsentiert, wurde darauf geachtet, die darin angezeigten Daten möglichst direkt aus den Originalquellen zu beziehen.

6.2.1.1. Liste aller API-Methoden

Die Liste der verfügbaren Methoden wird direkt aus dem Programmcode abgeleitet. Dazu wird mit Hilfe von Reflections⁵ in einem ersten Schritt die Liste aller Controller-Klassen eruiert und in einem zweiten Schritt deren Methoden extrahiert, die einen API-Aufruf repräsentieren.

⁵<http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/package-summary.html>

6.2.1.2. HTTP-Verb und Pfad

HTTP-Verb und Ressourcenpfad werden aus der «routes»-Datei geladen. Auch das Serverframework (Play Framework) lädt diese Informationen aus dieser Datei zur Delegation von Anfragen von Clients an den richtigen Controller. Damit ist sichergestellt, dass diese Daten in der API-Dokumentation stets aktuell sind.

6.2.1.3. Beschreibungen

Die verschiedenen Beschreibungen der Methoden werden aus Annotationen der Controller-Methode generiert. In Abbildung 6.2 ist ein Beispiel eines Controllers mit Annotationen abgebildet. Die Annotationen beschreiben:

- Alle Parameter
- Die Methode im Ganzen
- Dlle möglichen Rückgabestati und deren konkrete Bedeutung
- Ob eine Authentifizierung nötig ist
- Beispielaufrufe (siehe 6.2.2)

```
@Transactional()
@GuaranteeAuthenticatedUser
@QueryParameters({
    @Parameter(name = "id", isId = true, format = Long.class, description = "The id of the ..."),
    @Parameter(name = "name", description = "the new name of the new DKS to update"),
    @Parameter(name = "url", description = "the URL where the DKS is")
})
@QueryDescription("Updates a DKS (Decision Knowledge System).")
@QueryResponses({
    @QueryResponses.Response(status = NOT_FOUND, description = "If no entity with the given..."),
    @QueryResponses.Response(status = BAD_REQUEST, description = "If the request parameter ..."),
    @QueryResponses.Response(status = OK, description = "The new created entity is returned")
})
@QueryExamples({
    @Example(id = "9999", parameters = {"Example DKS", "http://a-dks.com"}),
    @Example(id = "REFERENCE_DKS_100000000000000073", parameters = {"Example DKS", "http:..."}),
    @Example(id = "REFERENCE_DKS_100000000000000074", parameters = {"Example DKS", ""})
})
public Result update(long id) {
    return update(DKS_DAO, DKS_LOGIC, DecisionKnowledgeSystem.class, id);
}
```

Abbildung 6.2.: Annotations im DecisionKnowledgeSystemController

6.2.2. Beispielaufrufe

Um dem Benutzer anzeigen zu können, dass und wie die Methode wirklich funktioniert, werden für jede Methode Beispielaufrufe und deren Antworten live generiert und angezeigt. Dazu werden bei jeder Generierung der API-Dokumentation zuerst einige Beispieldaten erstellt und anschliessend auf diese Daten zugegriffen über die Methoden, analog einem externen, unabhängigen Client. Die erhaltenen Ergebnisse werden dann in der

API als «The previous command **did** return» angezeigt. Falls eine solche Simulation, beispielsweise aufgrund von externen Abhängigkeiten, nicht möglich ist, so ist die vom Entwickler erwartete Antwort auch in der Annotation definiert und wird dem Benutzer als «The previous command **would probably** return» angezeigt.

Damit die dafür generierten Beispieldaten nicht mit den echten Daten des Systems interferieren, existiert für diesen Teil eine separate Datenbank. Diese muss der Benutzer jedoch nicht konfigurieren. Da die Daten nur während der Generation der API-Dokumentation benötigt werden und nicht längerfristig persistiert werden müssen, werden sie lediglich in einer SQLite-Datenbank⁶ gespeichert.

6.3. Client

Die EEPPI Clientapplikation nutzt die RESTfull Schnittstelle zum Datenaustausch mit dem Server sowie zur Kommunikation mit Remote-Hosts über den Proxy (Cross-Originale Aufruf).

Um auf dem Client einfach und flexibel Prototypen aus übertragenen JSON-Objekten instanzieren zu können, gibt es eine Object Factory. Diese baut anhand einer Factory-Konfiguration, die jedes übertragbare Objekt deklarieren muss, Objekte zusammen und füllt sie mit Daten. Diese zusätzliche Konfiguration ist notwendig, da JavaScript nicht genügend Typeninformationen besitzt, aus denen sich die erforderlichen Informationen ermitteln lassen und TypeScript diese nicht automatisch generieren kann. Alternativ liesse sich die Prototypeninstanziierung durch Factoryfunktionen pro Objekt umsetzen, dies hat sich jedoch während der Entwicklung als wartungsintensiv und Duplicated-Code-lastig erwiesen.

6.4. HTTP Verben

RESTfull impliziert die Verwendung der richtigen Verwendung der HTTP Verben:

GET für Abfragen

POST für Create-Operationen

DELETE für Lösch-Operationen

PUT oder POST für Update-Operationen.

Viele Netzwerkadministratoren erlauben PUT und DELETE nicht und blockieren entsprechenden Verkehr. Aus diesem Grund setzen Entwickler häufig nur GET und POST ein.

Wir haben uns entschieden, serverseitig beides zu ermöglichen: Um Objekte zu löschen kann entweder 'DELETE /<entity>' oder 'POST /<entity>/delete' aufgerufen werden.

Die Client Applikation verwendet nur POST und GET, dies kann allerdings umkonfiguriert werden. PUT Requests werden von EEPPI in der Standardkonfiguration gar nicht verwendet, es kommt stattdessen immer POST zum Einsatz.

⁶Einfache Datenbank, in welcher die Daten in einer einzigen Datei gespeichert werden

7. Benutzeroberfläche

7.1. Userinterface-Mocking

Für einen Erstentwurf des Userinterface hat das Team ein Wireframe-Brainstorming durchgeführt. Dazu hat jedes Teammitglied Wireframes und Workflows entworfen.

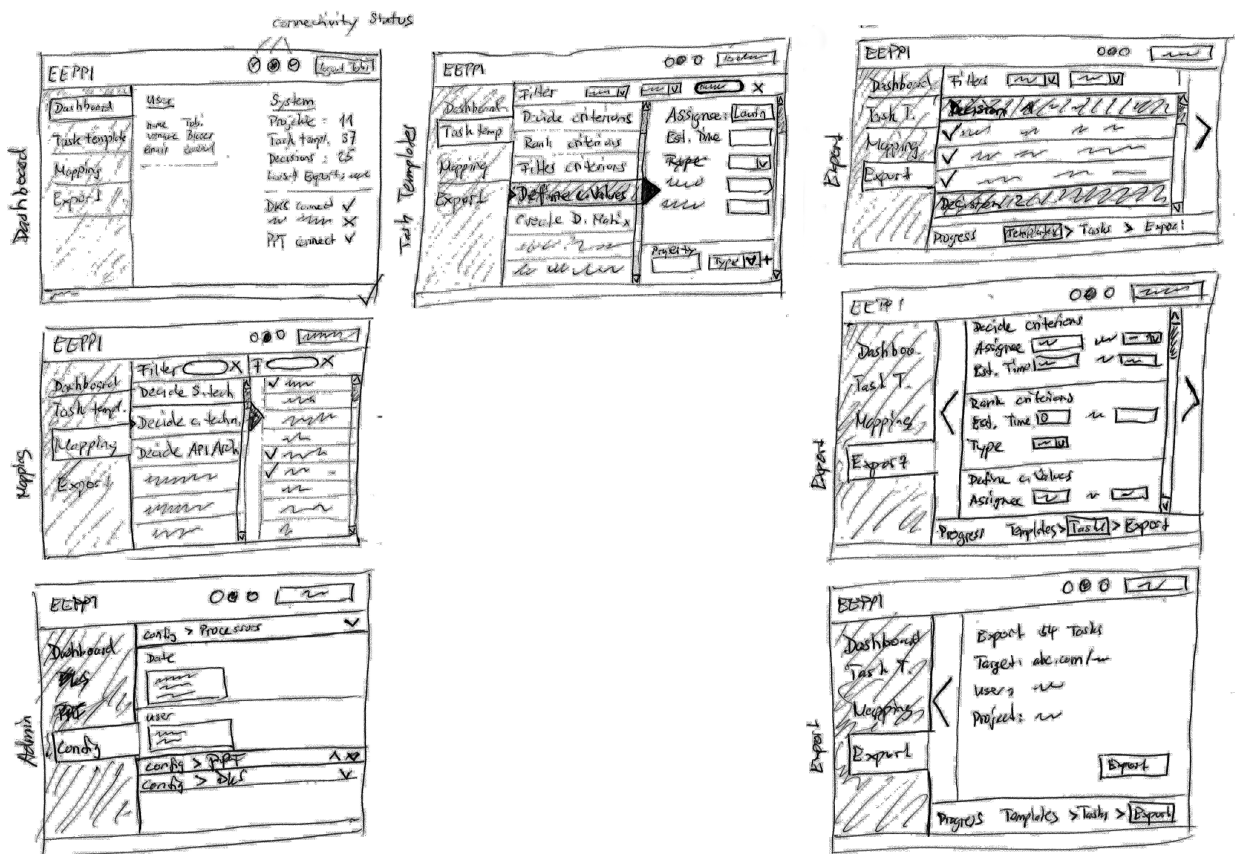
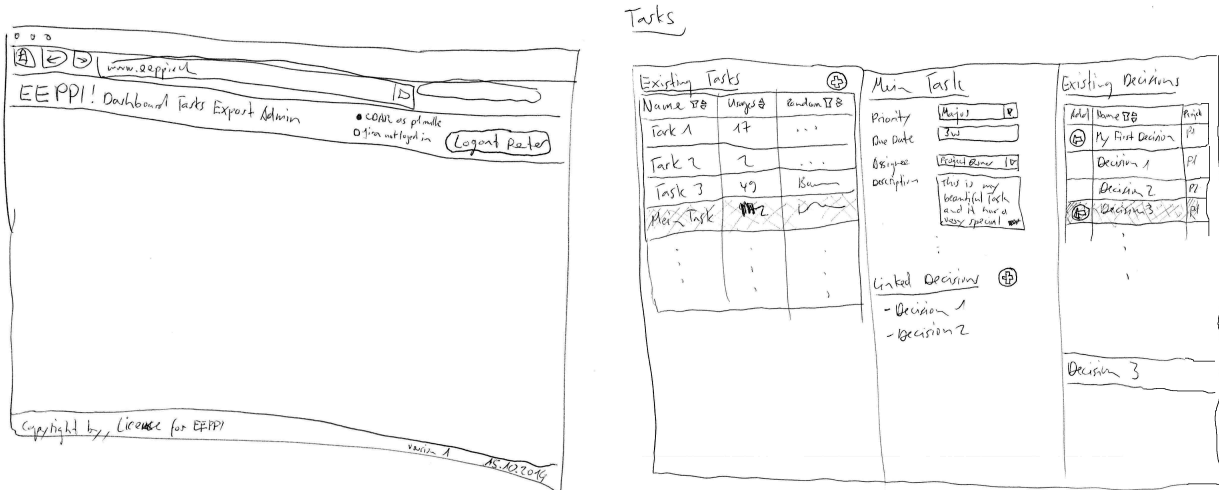
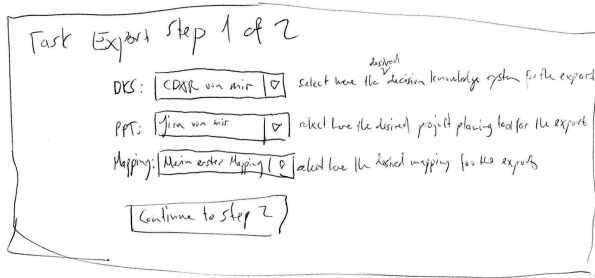


Abbildung 7.1.: Wireframes Tobias



Export Step 1



Export Step 2

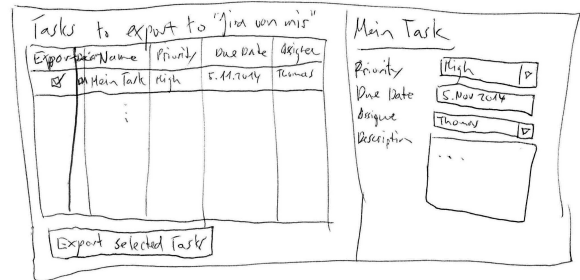


Abbildung 7.2.: Wireframes Laurin

Anschliessend wurden diese Entwürfe gemeinsam gesichtet, Wireframes aussortiert oder ausgewählt und Ideen zu neuen Wireframes kombiniert.

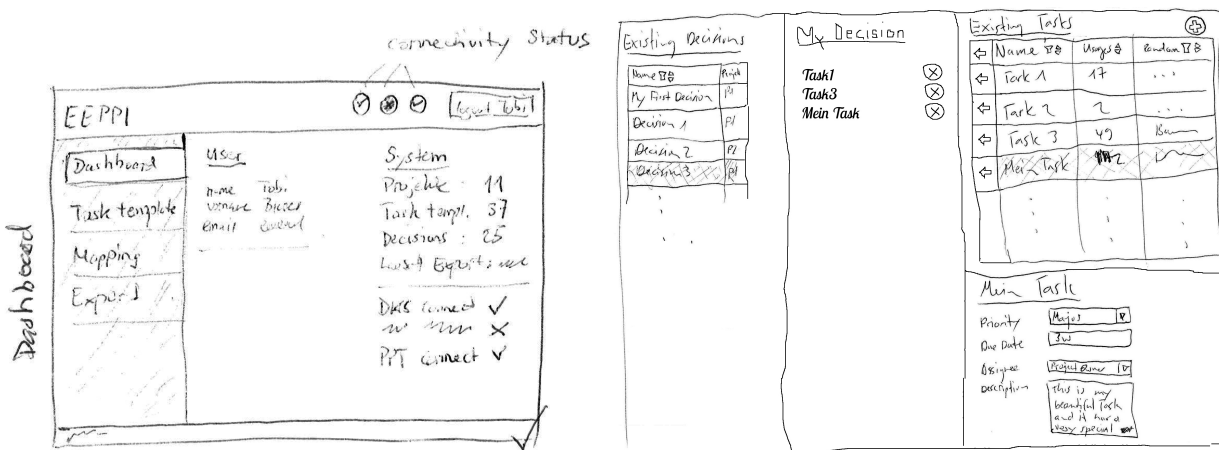


Abbildung 7.3.: Dashboard & Tasks

Für den Taskexport fiel die Entscheidung auf einen Assistent mit drei Schritten:

1. Auswählen von Projekt und Entscheidungswissenssystem sowie des Mapping sets

2. Bearbeiten der zu exportierenden Tasks, entfernen von nicht gewünschten
3. Auswahl des Projektplanungstools, exportieren sowie Übersicht über den Status eines laufenden Exports

Eine Progressbar im unteren Bereich soll dem Benutzer jederzeit anzeigen, bei welchem Schritt er sich befindet.

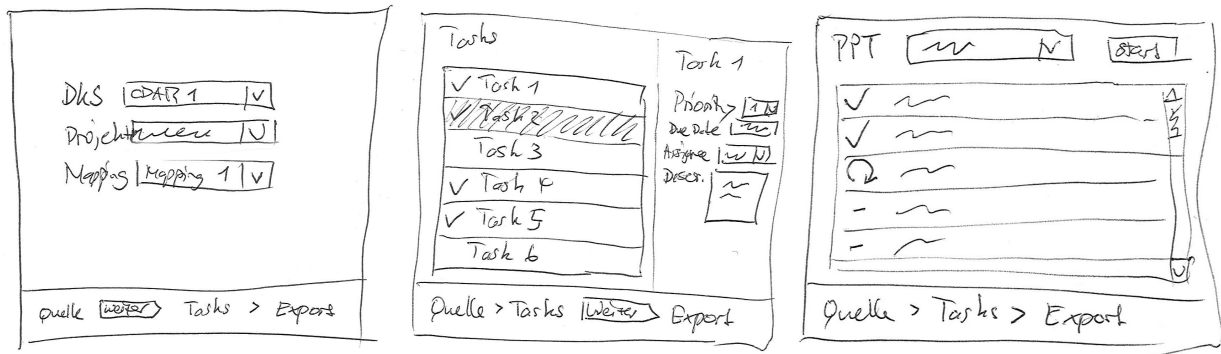


Abbildung 7.4.: Export Assistent

Der Benutzer kann nur weitergehen, nicht jedoch zurück, da dies durch den Umstand, dass aus Tasktemplates Tasks generiert werden, zu Datenverlust führen könnte.

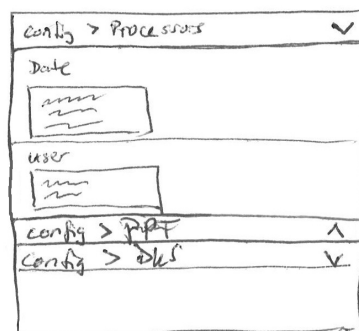


Abbildung 7.5.: Export Assistent

Der Administrationsbereich setzt sich vorwiegend aus einer aufklappbaren Liste zusammen (Accordion). Dadurch wird dem Administration eine gute Übersicht und einen schnellen Zugriff gewährleistet.

Die Navigation soll entweder unterhalb des Headers oder auf der linken Seite angebracht werden. Dazu soll während der Umsetzung überprüft werden, ob auf der linken Seite genügend Platz vorhanden ist, wenn die Mapping Ansicht geöffnet ist.

7.2. Finales Userinterface

Anhand der Mockups und entwickelten Workflows wurde anschliessend das finale Interface umgesetzt. Dabei wurden iterativ Bereiche angepasst und verbessert. Dies war

insbesondere in der Darstellung der Problems & Tasktemplates notwendig, da wesentlich mehr Daten visualisiert werden sollten, als ursprünglich im Mockup vorgesehen.

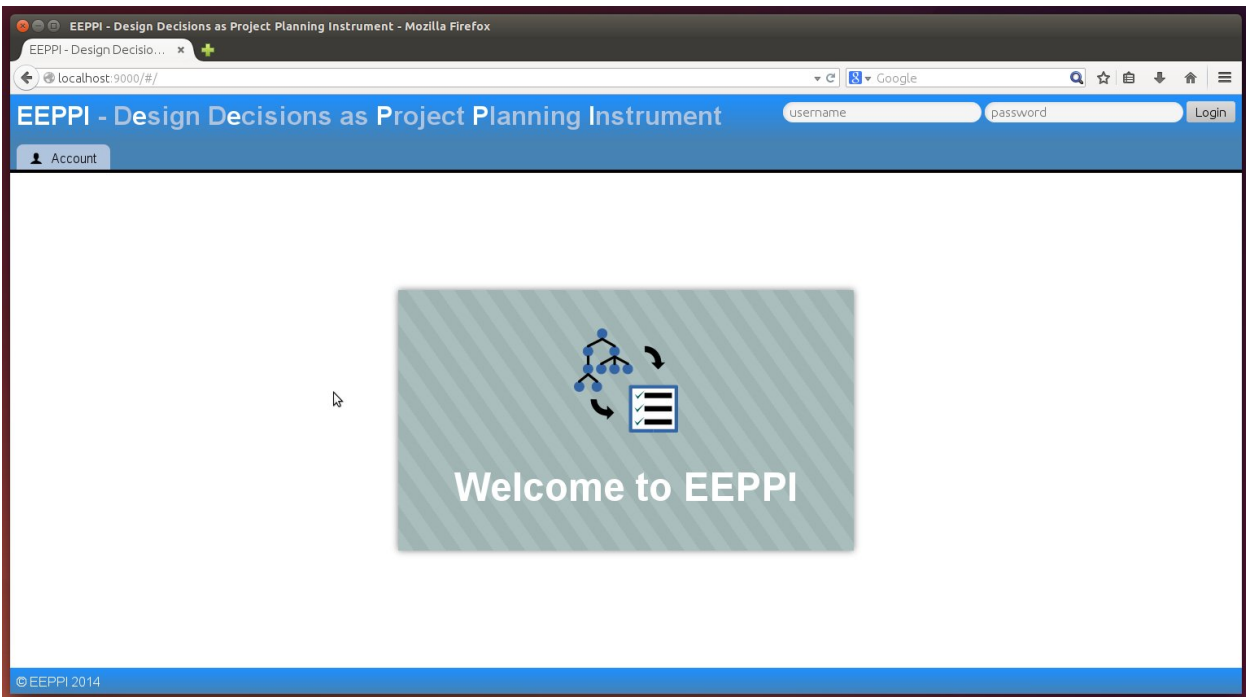


Abbildung 7.6.: EEPPi Home Screen im Browser

Auf die Umsetzung des Dashboard wurde verzichtet, da viele der dafür notwendigen Informationen im tief priorisierten und darum nicht umgesetzten Feature «Inform user about network and system status» enthalten waren und entsprechend nicht verfügbar waren. Anstelle wurde ein einfacher Home Screen implementiert (siehe Abbildung 7.6).

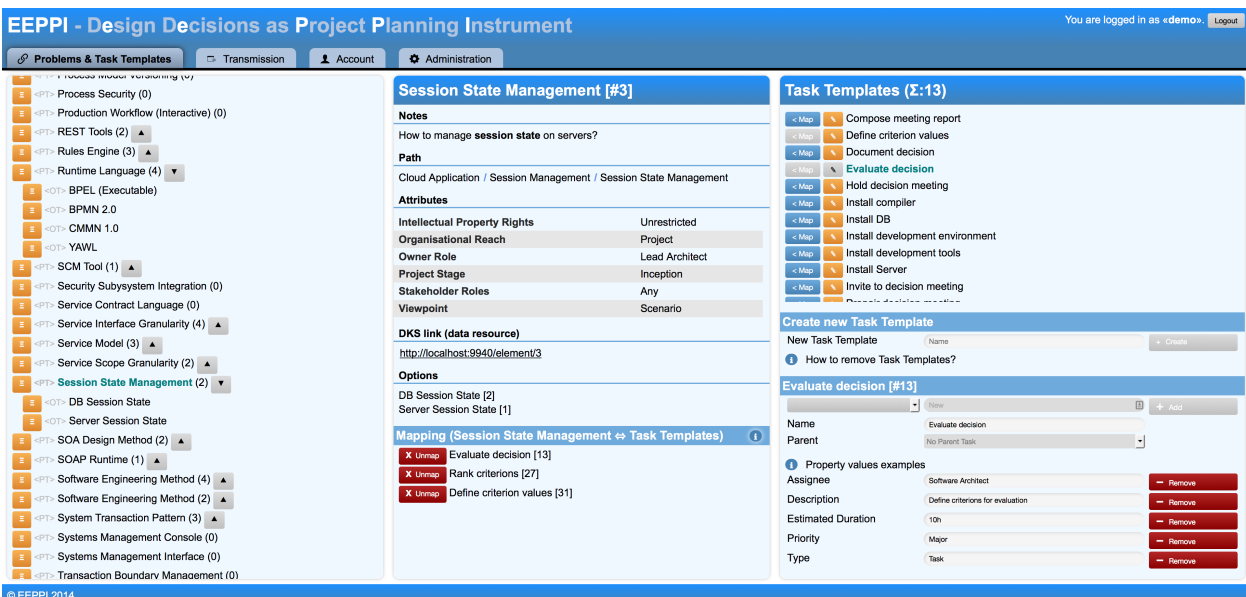


Abbildung 7.7.: Problems & Tasktemplates

Die Ansicht der verfügbaren Problems, der Tasktemplates und deren Mapping wurde nach dem Mockup umgesetzt und anschliessend noch um viele zusätzliche Informationen erweitert (Abbildung 7.7). So kam beispielsweise zur Detailansicht der Mappings noch eine Detailansicht des ausgewählten Problems hinzu. Damit kann sich der Benutzer eine bessere Übersicht darüber verschaffen, ob er das richtige Problem mappt. Insbesondere bei ähnlichen Namen der Problems ist dies von Vorteil. In diesen Bereich wurde auch eine HTML-Unterstützung für Notes eingebaut, sodass diese mit HTML-Tags ausgezeichnet werden können.

EEPPi - Design Decisions as Project Planning Instrument You are logged in as «hmeyer» Logout

Problems & Task Templates Transmission Account Administration

Administration

Manage properties, systems and requests.

Manage Decision Knowledge Systems (DKS)

A DKS is the source of all your Problems and Options. It is only read, nothing is written to it. In this version of EEPPi only one DKS is supported, however you can change it's URL here.

CorporationDKS CorporationDKS eeppi.corporation.com

Manage Task Template Properties

Task Template Properties are the different properties a Task Template can possess. Task Templates always have a name and you can enrich them here with additional Properties.

New Property Name + Add

New Name

How to remove Task Template Properties?

Manage Projects

You can group your work into different Projects and provide access to these to different people. You could manage them here. However, in this version of EEPPi only one Project is supported and it can't be changed.

Manage Project Planning Tools (PPT)

EEPPi supports different PPTs. If you have more than one type of them (e.g. Jira and Redmine) you can configure different PPTs here. However, this is only the type not the concrete instance. Information for the concrete instance are configured in [PPT Accounts](#). However, in this version of EEPPi only one PPT is supported and it can't be changed.

© EEPPi 2014

Abbildung 7.8.: Administrationsbereich

Beim Administrationsbereich (Abbildung 7.8) haben wir uns nur sehr beschränkt an die Mockups gehalten, da wir bald gemerkt haben, dass die darzustellenden Informationen nicht mit dem Mockup zusammenpassen. Zum Zeitpunkt, als wir die Mockups erstellten, war noch nicht genau klar, welche Informationen in diesem Bereich untergebracht werden sollen.

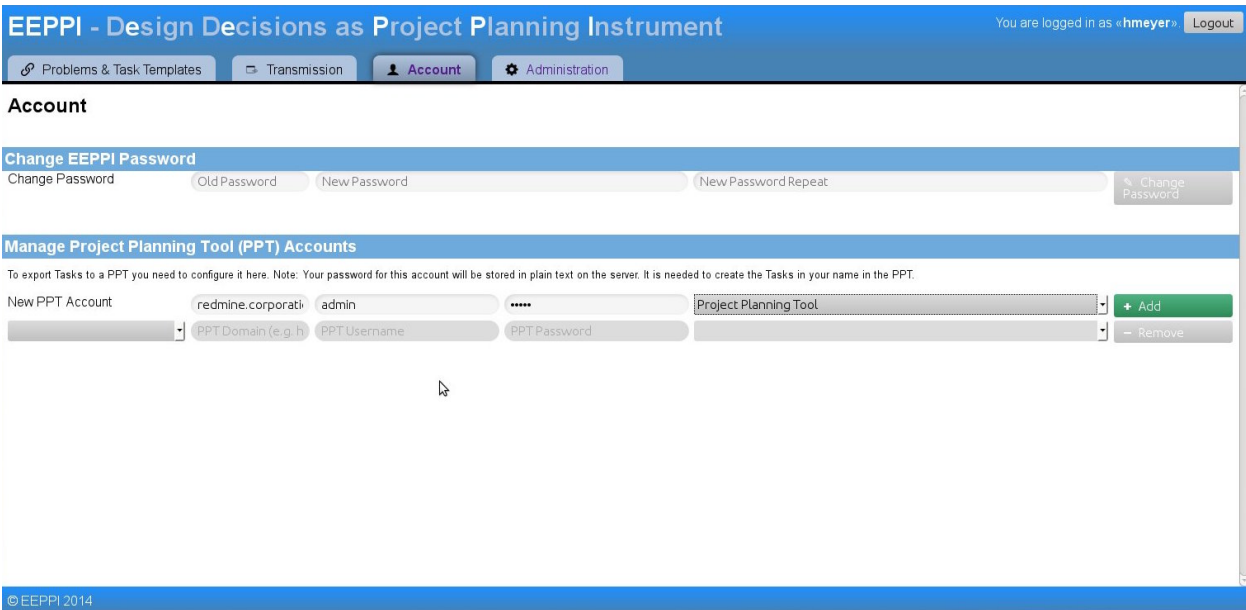


Abbildung 7.9.: Accountverwaltung

Für den Accountverwaltungsbereich (Abbildung 7.9) gab es keine Mockups, da zum damaligen Zeitpunkt noch Unklarheit über das Usermanagement bestand. Entsprechend wurde dieses analog der Administration gestaltet.

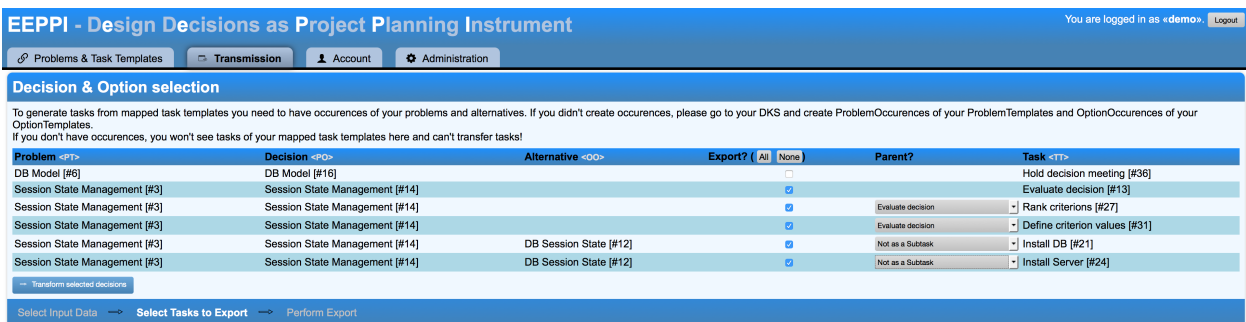


Abbildung 7.10.: Übertragen von Tasks an ein Projektplanungstool

Im Übertragungsbereich (Abbildung 7.10) ist gegenüber den Mockups die Editierfunktion für zu exportierende Tasks entfallen, dieses Feature wurde vom Betreuer, als Ansprechpartner der Kundengruppe, mit niedriger Priorität eingestuft.

8. Lizenzen und verwendete externe Produkte

8.1. EEPPI

EEPPI und die zugehörige Dokumentation stehen unter den folgenden Lizenzen:

EEPPI Source Code	Apache 2
Bachelorarbeit «Entwurfsentscheidungen als Projektplanungsinstrument»	CC 4.0

8.2. Verwendete Frameworks und Libraries

Um nicht von Grund auf alle Komponenten selbst programmieren zu müssen und um auf die Erkenntnisse anderer Entwickler aufbauen zu können, haben wir einige bestehende Frameworks verwendet. Diese sind nachfolgend, und die dazugehörigen Lizenzen in Absatz 8.4 aufgeführt.

Verwendung	Framework/Library	Version	URL	Lizenz
Framework (Client)	AngularJS	1.3.0	https://angularjs.org/	MIT License
Framework (Server)	Play Framework	2.3.6	https://www.playframework.com/	Apache 2
Library (Server)	Hibernate Entitymanager	4.3.6.Final	http://hibernate.org/orm/	LGPL
Library (Server)	PostgreSQL Driver	9.1-901.jdbc4	http://mvnrepository.com/artifact/org.postgresql/postgresql	PostgreSQL
Code Library (Server)	Jetbrains Annotations	7.0.2	http://mvnrepository.com/artifact/com.intellij/annotations	Apache 2
Test Framework (Client)	Jasmine	2.0	http://jasmine.github.io/	MIT
Test Library (Server)	Mockito	1.10.8	https://code.google.com/p/mockito/	MIT
Test Library (Server)	PowerMock	1.5.6	https://code.google.com/p/powermock/	Apache 2
Test Library (Server)	Selenium	2.43.1	http://www.seleniumhq.org/	Apache 2

8.3. Verwendete Tools und Technologien

Verwendung	Tool/Technologie	URL	Lizenz
Client	TypeScript	http://www.typescriptlang.org/	Apache2
Client Dokumentation	Typedoc	https://www.npmjs.com/package/typedoc	Apache 2
Client & Server	Less	http://lesscss.org/	Apache 2
Server	Java	http://www.oracle.com/technetwork/java/	GPL, Java Community Process
Projekt	Vagrant	https://www.vagrantup.com/	MIT
Projekt	Virtualbox	https://www.virtualbox.org/	GPL

8.4. Lizenzen

Kurzname	Voller Name	URL	Bedingung ¹	Erlaubt ¹	Verboten ¹
Apache 2	Apache 2 License	http://www.apache.org/licenses/LICENSE-2.0	<ul style="list-style-type: none"> • Lizenz und Copyright Informationen beilegen, • Änderungshistorie angeben 	<ul style="list-style-type: none"> • Kommerzielle Nutzung, • Vertrieb, • Veränderung, • Patent Erteilung, • Privater Gebrauch, • Weitere Lizenz 	<ul style="list-style-type: none"> • Haftbar machen, • Markenkennzeichen verwenden
CC 4.0	Creative Commons Attribution 4.0 International License	http://creativecommons.org/licenses/by/4.0/	<ul style="list-style-type: none"> • Lizenz und Copyright Informationen beilegen 	<ul style="list-style-type: none"> • Kommerzielle Nutzung, • Kopieren und weitergeben in beliebigem Format und Medium • Kombinieren, verändern und darauf aufbauen 	<ul style="list-style-type: none"> • Originalen Urheber unterschlagen
GPL	GNU General Public License 2.0	http://www.gnu.org/licenses/gpl-2.0.html	<ul style="list-style-type: none"> • Source öffentlich, • Lizenz und Copyright Hinweise, • Änderungshistorie angeben 	<ul style="list-style-type: none"> • Kommerzielle Nutzung, • Vertrieb, • Veränderung, • Patent Erteilung, • Privater Gebrauch 	<ul style="list-style-type: none"> • Haftbar machen, • Weitere Lizenz

¹Quelle: gemäss <http://choosealicense.com/> [3]

Kurzname	Voller Name	URL	Bedingung ¹	Erlaubt ¹	Verboten ¹
LGPL	GNU Lesser General Public License	https://www.gnu.org/licenses/lgpl.html	<ul style="list-style-type: none"> • Quellcode öffentlich, • Library Verwendung, • Lizenz und Copyright Informationen beilegen 	<ul style="list-style-type: none"> • Kommerzielle Nutzung, • Vertrieb, • Veränderung, • Patent Erteilung, • Privater Gebrauch, • Weitere Lizenz 	<ul style="list-style-type: none"> • Haftbar machen
MIT	MIT License	http://opensource.org/licenses/MIT	<ul style="list-style-type: none"> • Lizenz und Copyright Informationen beilegen 	<ul style="list-style-type: none"> • Kommerzielle Nutzung, • Vertrieb, • Veränderung, • Privater Gebrauch, • Weitere Lizenz 	<ul style="list-style-type: none"> • Haftbar machen
PostgreSQL	PostgreSQL License	http://opensource.org/licenses/postgresql	ähnlich wie die BSD oder MIT Lizenz (siehe oben)	ähnlich wie die MIT Lizenz (siehe oben)	ähnlich wie die MIT Lizenz (siehe oben)

9. Ergebnisse

9.1. Zielerreichung

Die Zielsetzung von EEPPI lautet in der Aufgabenstellung wie folgt:

«Ziel für den Kunden ist es, aus noch offenen und aus bereits getroffenen Architekturentscheidungen Aufgaben (Tasks) abzuleiten und in eine Taskmanagementsoftware zu überführen, um diese anschliessend in dieser Software verwalten zu können. In dieser Arbeit sollen das Mapping-Konzept und die Tool-Architektur entworfen sowie eine Implementierung in Form eines Tools erstellt werden.

Das Mapping-Konzept beinhaltet die Art der Abbildung von Entscheidungen aus dem CDAR-Tool auf Tasks eines Projektplanungstools. Die zu entwerfende Toolarchitektur zeigt den konkreten Aufbau einer solchen Applikation.»

Aus dieser Zielsetzung leiten sich drei Fragen ab:

1. «Wie lässt sich Entscheidungsbedarf in Form von agilen Planungselementen darstellen?»
2. «Welche Umsetzungstasks ergeben sich aus getroffenen Entscheidungen?»
3. «Wie können die Metamodelle und Werkzeugschnittstellen der beiden Domänen Architekturentscheidungen und Projektplanung aufeinander abgebildet und miteinander integriert werden?»

9.1.1. Hauptziel

Hauptziel der Arbeit war es, für das Ableiten von Tasks basierend auf Architekturentscheidungen, die folgenden Teilziele zu erfüllen:

1. Ein Mapping-Konzept zu erstellen
2. Eine Tool-Architektur zu entwerfen
3. Eine Implementierung in Form eines Tools zu erstellen

Das Mapping Konzept wurde im Rahmen der Konzeptphase zusammen mit der Tool-Architektur erarbeitet. Diese beiden erfolgreich erreichten Ziele waren Voraussetzung für das erfolgreiche Abschliessen des dritten Ziels, der Implementierung von EEPPI.

9.1.2. Rückblick auf die Fragestellungen der Aufgabenstellung

9.1.2.1. «Wie lässt sich Entscheidungsbedarf in Form von agilen Planungselementen darstellen?»

Im Allgemeinen kann gesagt werden, dass auch das Treffen einer Entscheidung als Task in einem Projektplanungstool repräsentiert werden kann. Ein Benutzer kann den Entscheidungsbedarf in Form eines oder mehrerer Tasktemplates in EEPPI modellieren, anschliessend mit der Entscheidung verknüpfen und ins Projektplanungstool übertragen.

Je nach Grösse, Komplexität und Wichtigkeit der Entscheidung kann der Benutzer dafür einen kleinen Task (wie beispielsweise «Entscheid treffen») oder auch mehrere grosse Tasks erstellen (wie beispielsweise «Entscheidungssitzung abhalten» oder «Alternativen evaluieren») und diese gegebenenfalls hierarchisch verknüpfen (Subtasks).

Zusammengefasst stellt der Benutzer Entscheidungsbedarf in Form von agilen Planungselementen dar, indem er diese in EEPPI modelliert und in ein Projektplanungstool übertragen lässt, wo sie verwaltet und weiterverarbeitet werden können.

9.1.2.2. «Welche Umsetzungstasks ergeben sich aus getroffenen Entscheidungen?»

Dies hängt sehr stark von der entsprechenden Entscheidung ab. Je nach Projekt können durchaus generische Tasks, wie beispielsweise «Entscheid dokumentieren» oder «Entscheid kommunizieren» vorkommen, doch die meisten Umsetzungstasks sind stark entscheidungsspezifisch.

Für ähnliche Projekte werden sich ähnliche Tasks-Muster ergeben, die Wiederverwendbarkeit wird hoch sein. Für neue und von bisherigen stark unterschiedliche Projekte wird der Benutzer viele neue Tasktemplates benötigen, entsprechend präsentieren sich auch die erzeugten Tasks.

Eine Gruppierung oder Kategorisierung von Tasks zur einfacheren Wiederverwendbarkeit ermöglicht das Arbeiten mit thematischen Taskgruppen. Beispielsweise könnte «Entscheid kommunizieren» aus den Aufgaben «Entscheid publizieren» und «Betroffene Informieren» bestehen. Insbesondere für Entscheidungstasks, wie die oben genannten, ist dies sinnvoll. Um Entscheidungen zu treffen, existieren weniger mögliche Aufgaben, als bei der Umsetzung einer getroffenen Entscheidung. Somit lassen sich die Tasktemplates häufiger wiederverwenden als bei Umsetzungstasks.

9.1.2.3. «Wie können die Metamodelle und Werkzeugschnittstellen der beiden Domänen Architekturentscheidungen und Projektplanung aufeinander abgebildet und miteinander integriert werden?»

Der kleinste gemeinsame Nenner der drei Domänen - Entscheidungswissensverwaltung, Projektplanung und die EEPPI-Domäne - stellen die Schnittstellen dar. Sie bilden auf jeder Seite einen Teil der Domäne ab und ermöglichen das Integrieren eines kleinen Teils einer anderen Domäne. Über diesen Korridor werden Daten ausgetauscht und für die Werkzeuge bereitgestellt.

Konkret bindet EEPPI sowohl Entscheidungswissenssysteme als auch Projektplanungstools als externe Systeme an und ermöglicht dem Benutzer, EEPPI sehr flexibel an die

beiden andern Domänen anzupassen.

Modellmässig stellt EEPPI sehr wenige Anforderungen an beiden Domänen. Konkret muss ein Architekturstool folgende Eigenschaften aufweisen:

- Unterscheidung zwischen Vorlagen und konkreten Elementen
- Unterscheidung zwischen Eltern- (zum Beispiel Problemen) und Kinder-Elementen (zum Beispiel Optionen)
- Werkzeuge zur Auflistung aller Elemente und zur Filterung

Ein Projektplanungstool muss im Prinzip lediglich einzelne Elemente (zum Beispiel Tasks) erstellen können.

Konkrete Anforderungen an die Elemente der beiden Domänen gibt es nicht, es ist aber für eine effiziente Verwendung förderlich, wenn sie eine Vielzahl von gleichen oder zumindest ähnlichen Attributen aufweisen. Sind die Schnittstellen stark unterschiedlich, sind viele Processoren notwendig, um die Daten zu konvertieren.

9.1.3. Erfolgsfaktoren

In der Aufgabenstellung wurden kritische Erfolgsfaktoren festgelegt. Diese werden im folgenden mit Rückblick auf das Projekt beleuchtet.

9.1.3.1. «Niedrige Einstiegshürden für User»

Dieser Erfolgsfaktor setzt sich aus drei Teilaspekten zusammen.

Der erste Aspekt ist ein geringer Installationsaufwand. Die Installation von EEPPI ist so einfach wie die Installation eines Jira und in weniger als einer Stunde erledigt. Für Benutzer wird das empfohlene Vorgehen im Abschnitt D genau beschrieben.

Ein weiterer Aspekt sind Lizenzfragen. Eine die zukünftige Nutzung stark einschränkende Lizenz würde viele potentielle Interessenten abschrecken. Aus diesem Grund steht EEPPI, wie in Abschnitt 8.1 beschrieben, unter der sehr offenen Apache 2 Lizenz.

Letzter Aspekt ist die Robustheit im Betrieb. EEPPI war als Forschungsprojekt nicht längere Zeit im Produktivbetrieb und konnte sich deshalb damit nicht beweisen. Zur Verbesserung der Robustheit hat das Projektteam darum als letzte Iteration während der Entwicklung eine Phase zur Stabilisierung und Fehlerkorrektur durchgeführt.

9.1.3.2. «Modularität und Erweiterbarkeit»

Dieser Punkt beleuchtet die Schnittstellen und deren Dokumentation von EEPPI. Schnittstellen bietet, wie in der Architektur beschrieben, lediglich der Server und auch nur der Server verwendet externe Schnittstellen. Der Client verwendet einzig die Schnittstelle des Servers und zeigt damit gleich, dass die Schnittstelle des Servers die notwendige Funktionalität anbietet. Wie in Abschnitt E.1 erläutert, gibt es eine ausführliche Dokumentation zur Schnittstelle. Der grundlegende Aufbau von EEPPI ist mit diesem Dokument hier dokumentiert und kann als Basis für Folgearbeiten verwendet werden.

9.1.3.3. «Reife der Konzepte»

Dieser Erfolgsfaktor widmet sich der Konfigurierbarkeit, der Flexibilität und der Eleganz der Mapping-Konzepte von EEPPI. Bei der Entwicklung von EEPPI hat das Projektteam grossen Wert auf genau diese Punkte gelegt. Sehr viele Parameter lassen sich konfigurieren und auch die grundlegende Funktion, das Übertragen von Tasks an ein Projektplanungstool, ist direkt durch den Benutzer im Frontend auf hohem Level anpassbar. Das Mapping wurde so ausgelegt, dass der Benutzer nicht direkt Tasks erstellen muss, sondern dass er in Form eines Metamappings Tasktemplates erstellt, aus denen bei einer Übertragung in ein Projektplanungstool dann Tasks erstellt werden.

9.2. Umsetzung der Userstories

Im Rahmen der Konzeptphasen wurden Userstories definiert, unterteilt in Core- und Advanced-Userstories. Die Core-Userstories wurden alle im Rahmen der Arbeit umgesetzt.

Mit Ausnahme der User Stories, die auf tief priorisierten und darum nicht umgesetzten Features basieren, wurden auch alle Advanced-Userstories umgesetzt. Bei den nicht umgesetzten handelt es sich um die folgenden:

BA-40 Zugriffsrechte und Fähigkeiten

BA-41 Report Task Übertragung

Die User Story «BA-39 DKS¹ Space mit PPT² Projekt verknüpfen» wurde aus technischen Gründen anders umgesetzt, sodass Benutzern ein anderes Vorgehen anwenden müssen, um zum gleichen Ziel zu gelangen: Ein Benutzer kann nicht direkt DKS Spaces mit PPT Projekten über die Benutzeroberfläche verknüpfen, sondern muss vom Administrator einen Processor erstellen lassen, der den übermittelten DKS Space mit einem PPT-Identifizier verknüpft.

9.3. Offene Punkte, Bugs, technische Probleme

Aufgefundene Bugs wurden während der Stabilisierungsphase zwischen Feature- und Code-Freeze behoben. Nach dem Code-Freeze aufgefundene Bugs sind folgend aufgelistet.

Mehrfachregistrierung Wird mehrfach ein Benutzer mit gleichem Benutzernamen registriert, so wird vom Server die ID des bereits existierenden Benutzers zurückgegeben, anstelle einer Fehlermeldung über den bereits existierenden Benutzer. Existierende Benutzer werden jedoch nicht überschrieben oder verändert.

¹Entscheidungswissenssystem

²Projektplanungstool

9.4. Zusätzliche Features

Neben den zu Beginn geplanten Funktionen konnten während dem Projekt noch einige weitere Funktionen umgesetzt werden. Diese sind nachfolgend beschrieben.

9.4.1. API-Dokumentation

Zur Dokumentation der API Schnittstelle hat das Projektteam einen Mechanismus implementiert, der die Informationen über die Schnittstelle mittels Reflection aus dem Code und aus der Ressourcenkonfiguration des Play-Frameworks ausliest. Zusätzlich werden während der Erzeugung der Dokumentation konkrete Entitäten erstellt und reale API-Aufrufe gefahren. Die entsprechenden Live-Resultate werden als Aufrufresultate in der Dokumentation angezeigt. Dadurch erhält ein Leser realistische Benutzungsbeispiele für die API mit passenden Antwortdaten.

9.4.2. Strukturierung von Tasktemplates

Eine Strukturierung von Tasktemplates wurde zugunsten höher priorisierten Features als optionales Feature deklariert. Dem Team ist es jedoch in der letzten Woche vor dem Featurefreeze noch gelungen, eine schlanke Umsetzung einer Hierarchisierung von Tasktemplates zu implementieren. Dies ermöglicht das Erstellen von Subtasktemplates und das Exportieren von Subtasks.

9.4.3. Im- und Export der Daten von EEPPI

Über die API von EEPPI lassen sich alle Daten des Systems, ausgenommen Benutzerpasswörtern, exportieren und importieren. Dies ermöglicht das Transferieren von Tasktemplates und Mappings zu einer andern EEPPI-Instanz.

Zusätzlich lassen sich alle Daten auch durch ein Backup der Datenbank sichern und transferieren, da EEPPI keine Daten ausserhalb der im Play-Framework konfigurierten Persistenz³ ablegt.

9.4.4. Beispielrequest für Redmine

Am Anfang des Projektes wurde von Projektteam und Betreuer beschlossen, dass EEPPI grundsätzlich an jedes Projektplanungstool angebunden werden können soll, das die technischen Voraussetzungen erfüllt. Für die Entwicklung soll aber der Fokus vor allem auf einem einzigen System liegen, um sich nicht auf zu viele verschiedene APIs konzentrieren zu müssen. Das Projektteam hat zusätzlich zur Beispielkonfiguration für Jira und einer entsprechenden Demoinstallation auch eine Konfiguration für Redmine erstellt, eine Beispielinstallation aufgebaut und dessen Funktion demonstriert.

³Play legt die Daten je nach Konfiguration in einer Datenbank oder einer Datei ab. Im Falle der Datenbank können Backups mit deren Tools erstellt werden, im Falle der Datei (Infiledatenbank) die Datei selbst gesichert werden.

9.5. Erweiterungs-Möglichkeiten

EEPPI zeigt Möglichkeiten und Wege, wie Entscheidungsmanagement und Projektplanung zusammengebracht werden kann. Diese neuen Möglichkeiten wecken wiederum Begehrlichkeiten und Ideen. Entsteht das Bedürfnis, EEPPI zu erweitern, bieten sich verschiedene Möglichkeiten, die nachfolgend kurz angerissen werden.

9.5.1. Erweiterung der bestehenden Applikation

Die naheliegendste Erweiterungsmöglichkeit bietet die direkte Anpassung der Implementierung von EEPPI. Beschränken sich die Anpassungen nur auf die Funktionalität und nicht auf die Menge der persistierten Daten, so muss lediglich die Clientapplikation angepasst werden.

9.5.2. Ersatz der Serverapplikation

Die Serverapplikation verwaltet primär die Persistenz und stellt diese Funktionalität in Form der API zur Verfügung. Soll diese grundlegend verändert werden oder durch eine neue Technologie ersetzt werden, kann die Serverapplikation wie ein Modul ausgetauscht werden, ohne den Client anpassen zu müssen.

Die neue Serverapplikation muss lediglich den gleichen Funktionsumfang anbieten und die Daten in einem ähnlichen Aufbau liefern. Parameter wie zum Beispiel Ressourcenadressen lassen sich auf dem Client konfigurieren.

9.5.3. Ersatz der Clientapplikation

Die Clientapplikation besitzt den grössten Teil der Logik, so auch die Processors. Zudem ist sie verantwortlich für die Darstellung der Benutzeroberfläche. Die Benutzeroberfläche selbst kann einerseits durch Custom Styles⁴ sehr einfach umgestaltet werden, durch Anpassen der zu Grunde liegenden Templates sogar in sehr weitem Masse.

Soll die Clientapplikation jedoch komplett ersetzt werden, beispielsweise durch Ablösung einer neuen Technologie, so ist dies problemlos möglich. Die neue Applikation kann wie die bestehende Clientapplikation die Serverschnittstelle verwenden.

9.5.4. Erstellung eines Proxies

Da EEPPI eine geringe Kopplung zu den externen Systemen aufweist, ist es mit wenig Aufwand möglich, gewisse Funktionen in Form eines Proxies⁵ zu implementieren. Dabei würde EEPPI nicht verändert werden, sondern lediglich neue Referenzen zu den externen Systemen konfiguriert werden. Dieser Ansatz würde sich beispielsweise gut für die Implementierung von weiteren Projektplanungstool-Schnittstellen eignen (siehe 9.6.14).

⁴Dazu steht ein entsprechendes Stylesheet bereit, das nach den Default-Styles von EEPPI eingebunden wird und dazu gedacht ist, vom Integrator mit Organisations-spezifischen Styles ergänzt zu werden, um EEPPI in die Corporate Identity der Firma zu integrieren.

⁵Proxy-Pattern: <http://c2.com/cgi/wiki?ProxyPattern>

9.5.5. Verwendung der API

Einige mögliche neue Funktionen basieren auf Kernfunktionen von EEPPI, bieten aber einen weitergehenden Nutzen für den Benutzer (wie beispielsweise der Rückfluss von Informationen eines Tasks zurück in das Tasktemplate, siehe Abschnitt 9.6.12). Solche Funktionen könnten durch eine eigenständige Applikation implementiert werden, welche das API von EEPPI verwendet.

9.6. Mögliche Erweiterungen

EEPPI deckt die notwendige Kernfunktionalität ab, damit Benutzer angenehm arbeiten können. Im Rahmen einer Weiterentwicklung sind viele Möglichkeiten denkbar. Einige wie ein Rechte-System sind eher praktischer Natur, andere wie beispielsweise Vererbungsmöglichkeiten für Tasktemplates würden die Wiederverwendbarkeit verbessern und dem Benutzer einen echten Mehrwert bieten. Nachfolgend sind einige denkbare Erweiterungen von EEPPI erklärt.

9.6.1. Rechte und Rollen

Mit einem Rechte- und Rollen-Konzept könnte der Zugriff auf die Daten in EEPPI eingeschränkt werden und unterschiedlichen Zielgruppen unterschiedliche Datenstämme angeboten werden.

Zudem würde ein Rechte- und Rollen-Konzept die Möglichkeit der Mandantenfähigkeit bieten. EEPPI könnte als Cloud-Service angeboten werden und eine einzige Instanz könnte für mehrere Kunden verwendet werden.

9.6.2. Vererbende Tasktemplates

Tasktemplates, die Eigenschaften von andern Templates erben können, minimieren die Anzahl notwendiger Templates und erhöhen die Wiederverwendbarkeit.

Beispielsweise ist ein Tasktemplate für Sitzungen denkbar, mit davon abgeleiteten Templates für verschiedene Sitzungsarten. Dem Elterntemplate untergeordnete Subtasks wie zum Beispiel «Zur Sitzung einladen» oder «Sitzungs-Protokoll» versenden, würden automatisch auch auf Subtemplates angewandt, bzw. mit diesen ans Projektplanungstool übertragen.

9.6.3. Reporting

Eine weitere Erweiterungsmöglichkeit für EEPPI stellt eine Übersicht über die in das Projektplanungstool exportierten Tasks dar. Dies würde dem Benutzer den Umweg über das Projektplanungstool ersparen.

9.6.4. Import/Export

Damit der Benutzer die Daten auch in anderen Systemen verwenden kann, ohne dass dafür eine Anbindung an die API nötig wäre, könnte eine Möglichkeit zum Im- und Export der Daten implementiert werden. Die Daten könnten beispielsweise als CSV⁶- oder ganze Excel-Dateien importiert und exportiert werden.

9.6.5. Undo von übertragenen Tasks

Möglicherweise möchte ein Benutzer erstellte Tasks wieder zurückziehen, beziehungsweise wieder aus dem Projektplanungstool löschen. Sofern Projektplanungstools dies unterstützen, würde dies dem Benutzer eine Undo-Möglichkeit für fehlerhafte erstellte Tasks anbieten.

9.6.6. Bearbeitungsmöglichkeiten für zu übertragende Tasks

Unter Umständen möchte der Benutzer die Tasks nicht in der erstellten Rohform in das Projektplanungstool exportieren. Eine Bearbeitungsmöglichkeit vor dem Übertragen würde dem Benutzer eine nachträgliche Bearbeitung im Projektplanungstool ersparen. Aktuell muss er sie zuerst exportieren und dann direkt im Projektplanungstool anpassen.

9.6.7. Kaskadierende Processors

Die aktuelle Ausgabe von EEPPI unterstützt keine Processors innerhalb von Processors. Dem Benutzer würde das Verwenden von Processorwerten innerhalb eines Processors die Möglichkeit bieten, Processors wiederzuverwenden und generischer zu gestalten.

9.6.8. System Status Benachrichtigungen

Ein weiteres kleineres Feature stellt eine Anzeige über den Status der angebotenen Systeme dar. Denkbar in der Form eines einfachen Icons, das anzeigt, ob die Systeme (korrekt) konfiguriert sind, ob sie erreichbar sind und je nach System unter Umständen auch noch, ob sie korrekt arbeiten.

9.6.9. Verwendung mehrerer Entscheidungswissenssysteme

Aktuell unterstützt EEPPI nur ein Entscheidungswissenssystem⁷, welches konfiguriert werden kann. Durch die Integration mehrerer konfigurierbarer Entscheidungswissenssysteme als Datenquelle, wäre der Benutzer in der Lage, das gewünschte System für das Mapping oder als Quelle für die Erzeugung der Tasks auszuwählen.

⁶Comma Separated Values, einfaches Dateiformat zur Speicherung von strukturierten Daten

⁷Die technische Grundlage für mehrere Entscheidungswissenssysteme ist vorhanden, da sie als Entitäten abgelegt werden. Server wie Client verwenden jedoch nur eine einzige Defaultinstanz.

9.6.10. Verwendung mehrerer Projekte

Analog zum vorhergehenden Punkt unterstützt EEPPI auch nur ein einziges Projekt⁸. Die Möglichkeit mehrerer Projekte ermöglicht einerseits Mandantenfähigkeit in einer weiteren Dimension und andererseits auch Schutz des geistigen Eigentums, da zusammen mit dem Rechte-Rollen Konzept einzelnen Benutzern nur bestimmten Projekten zugeteilt werden könnten.

9.6.11. Unterscheidung von verschiedenen Projektplanungstools

Benutzer können aktuell in EEPPI Request-Templates sowie Projektplanungstool-Accounts einem Projektplanungstool-Typ zuordnen. Allerdings unterstützt EEPPI in der jetzigen Version nur einen einzigen Projektplanungstool-Typ(-Identifikator). Diesem sind auch alle Request-Templates und Accounts zugeordnet.

Eine Unterstützung für mehreren Projektplanungstool-Typ(-Identifikatoren) würde Benutzern Verwirrung ersparen, welchen Account sie jetzt für welches Request-Template verwenden können.

9.6.12. Rückfluss von Informationen des Tasks zurück in Tasktemplate

Wenn Benutzer einen Task aus einem Tasktemplate erstellen, wird dieser ins Projektplanungstool übertragen. Der Benutzer bearbeitet diesen dort und aktualisiert ihn während dem Projektverlauf. Die Erfahrungen, die dabei gemacht werden, bleiben im Task im Projektplanungstool und bringen weiteren Projekten keinen Gewinn.

Eine denkbare Erweiterungsmöglichkeit von EEPPI ist eine Rückführungsmöglichkeit von Erkenntnissen aus Projekten in die Tasktemplates zurück. Unter Umständen gäbe es gar die Möglichkeit, dies teilautomatisiert zu tun. Beispielsweise könnte man Änderungen der Tasks erkennen und dem Benutzer vorschlagen, diese ins EEPPI zu übertragen.

9.6.13. Rückfluss von Informationen ins Entscheidungswissenssystem

Einerseits ist der Rückfluss von statistischen Werten über die Verwendung von Entscheidungen denkbar, andererseits ein Rückfluss von Zuständen. Wird beispielsweise im Projektplanungstool der Task «Session State Evaluieren» geschlossen und «DB Session» als Option ausgewählt, so könnte diese Information im Entscheidungswissenssystem dazu dienen, die entsprechende Entscheidung als geschlossen zu markieren und mit der gewählten Option zu verknüpfen. Dem Benutzer würde diese Automatisierung Mehrspürigkeiten in verschiedenen Tools ersparen.

⁸Die technische Grundlage für mehrere Projekte ist vorhanden, da das Projekt eine Entität ist. Server wie Client verwenden jedoch nur eine einzige Defaultinstanz.

9.6.14. Weitere Projektplanungstool-Schnittstellen

Aktuell unterstützt EEPPI Projektplanungstools, die über eine JSON-Schnittstelle verfügen und eine Authentifizierung über HTTP-Basic-Authentication ermöglichen. Für diese Art der Authentifizierung muss Benutzername und Passwort im Klartext vorliegen.

Für erhöhte Sicherheit und Kompatibilität wäre eine Erweiterungsmöglichkeit von EEPPI denkbar, die Projektplanungstools auch über weitere Schnittstellen (beispielsweise XML) und weitere Authentifizierungs-Methoden (beispielsweise OAuth⁹) anbindet.

9.7. Zukunft von EEPPI

Abschliessend stellt sich die Frage, wie es mit EEPPI weiter geht. EEPPI verbindet Entscheidungs- und Projektmanagement und ist dadurch sehr von diesen Disziplinen abhängig. Projektplanungstools sind heute weit verbreitet, hingegen sind Entscheidungswissenssysteme noch nicht sehr populär. Aufgrund der Notwendigkeit eines solchen für EEPPI, ist es nach dem heutigen Stand unwahrscheinlich, dass sich EEPPI schnell weit verbreiten wird.

Das Team dieser Bachelorarbeit hat nicht vor, EEPPI selbst weiter zu entwickeln. EEPPI ist ein «Proof of Concept» und hat damit gut aufgezeigt, was möglich ist in diesem Bereich.

Wir hoffen, mit EEPPI dazu beigetragen zu haben, dass wenn sich auch Entscheidungswissenssysteme durchgesetzt haben werden, sich das Entscheidungs- und Projektmanagement stärker verbindet und sich damit Projekte besser und konsistenter planen, schätzen und umsetzen lassen.

Insbesondere für grosse Opensourceprojekte wie den Linux Kernel würde der Einsatz eines Entscheidungswissenssystems in Kombination mit EEPPI Sinn machen und zur verbesserten Nachvollziehbarkeit von Entscheidungen und daraus abgeleiteten Aufgaben führen.

Es würde uns insbesondere freuen, wenn viele Entscheidungen und die daraus resultierenden Tasks (ähnlich Wikipedia¹⁰ oder Stackoverflow¹¹) öffentlich zugänglich wären und damit einen Gewinn für die gesamte Gesellschaft bieten würden.

Fazit: Wir empfehlen als nächsten Schritt die Verbreitung von Entscheidungswissenssystemen zu fördern und nach Möglichkeit einen Standard und eine öffentliche, offene Entscheidungs-Bibliothek zu starten, ähnlich wie Wikipedia oder Stackoverflow.

⁹Ein offenes Authentifizierungs-Protokoll. OAuth beispielsweise würde Authentifizierung auch ohne das effektive Passwort ermöglichen: <http://oauth.net/2/>

¹⁰<http://www.wikipedia.org/>

¹¹<http://stackoverflow.com/>

10. Glossar

API Application Programming Interface, Schnittstelle eines Programms zum Austausch mit externen Anwendungen

CSS Cascading Style Sheets: Auszeichnungssprache für Webdokumente, um diese grafisch zu gestalten

CDAR Collaborative Decision Management and Architectural Refactoring Tool [12]

CORS Cross-Origin Resource Sharing: Teilen von REST-Ressourcen über mehrere Origins (Server)

DKS Entscheidungswissenssystem wie CDAR oder ADRepo

EEPPI Entwurfsentscheidungen als Projektplanungsinstrument

IFS Institut für Software, HSR Hochschule für Technik: <http://www.ifs.hsr.ch/>

Less Sprache, um CSS vereinfacht zu generieren

LOC Lines of Code: Anzahl Zeilen Code

Problem Space In Entscheidungswissenssystem abgelegtes Wissen, welches später über einen Solution Space hilft Projekte umzusetzen. (siehe auch Kapitel A.2.2)

Projektplanungstool Ein Tool, welches Tasks verwaltet. Es wird zum Planen und Durchführen von Projekten verwendet. (siehe auch Kapitel A.2.2)

REST Representational State Transfer: Art einer Schnittstelle von Webapplikationen. Jede Ressource hat eine eigene URL und kann mit der korrekten Verwendung der HTTP-Verben bearbeitet werden.
(siehe auch http://en.wikipedia.org/wiki/Representational_state_transfer)

Selenium Testframework, welches einen Browser startet und direkt darin die Webseite testet (siehe auch <http://www.seleniumhq.org/>)

SLOC Source Lines of Code: Total Anzahl Zeilen eines Codes, inklusive der irrelevanten Zeilen wie Leerzeilen

Solution Space Kopie eines Problem Spaces um konkret Entscheide für ein Projekt zu erstellen. (siehe auch Kapitel A.2.2)

Tasks Aufgabe, welche üblicherweise in einem Projektplanungstool abgelegt ist. (siehe auch Kapitel A.2.2)

Tasktemplate Vorlage im EEPPI für einen (konkreten) Task im Projektplanungstool

TDD Test Driven Development: Ansatz des Entwicklungsprozesses, bei welchem zuerst die Tests geschrieben werden und erst danach der entsprechende Code

Vagrant Virtualisierungsautomatisierungslösung von HashiCorp für verschiedene Virtualisierungsumgebungen: <http://www.vagrantup.com/>

Wissenskonsument Person, die Entscheidungswissenssysteme und EEPPI für die Umsetzung von einem Projekt verwendet. (siehe auch Kapitel A.2.2)

Wissensproduzent Person, die Wissen im Entscheidungswissenssystem/EEPPI erfasst. (siehe auch Kapitel A.2.2)

MVW/MV* Mode View Whatever: Fasst die Patterns MVC - Model View Controller, MVP - Model View Presenter, MVVM - Model View Viewmodel und ähnliche Patterns zusammen.

Literatur

- [1] Atlassian. *JIRA - Issue & Project Tracking Software*. 26. Sep. 2014. URL: <https://www.atlassian.com/software/jira> (besucht am 29.09.2014).
- [2] Vincent Driessen. *A successful Git branching model*. 5. Jan. 2010. URL: <http://nvie.com/posts/a-successful-git-branching-model/> (besucht am 29.09.2014).
- [3] GitHub. *Choosing an OSS license doesn't need to be scary - ChooseALicense.com*. 12. Dez. 2014. URL: <http://choosealicense.com/> (besucht am 12.12.2014).
- [4] Leonard Hand und Sabyasachi Biswas. "IBM Solution Design Method". 2008. URL: https://files.ifi.uzh.ch/rrerg/amadeus/teaching/courses/it_architekturen_hs10/Solution_Design_I.day.pdf (besucht am 10.01.2014).
- [5] Gregor Hope und Bobby Woolf. *Enterprise Integration Patterns - Pipes and Filters*. 2003. URL: <http://www.eaipatterns.com/PipesAndFilters.html> (besucht am 07.10.2014).
- [6] BuiltWith® Pty Ltd. *Framework technologies Web Usage Statistics*. 23. Sep. 2014. URL: <http://trends.builtwith.com/framework#> (besucht am 23.09.2014).
- [7] Olaf Prof. Dr. Zimmerman. "Architectural refactoring for cloud". Jahrestagung Architekturen 2014. Ladenburg, 2014. URL: http://www.ifs.hsr.ch/fileadmin/user/_upload/customers/ifs.hsr.ch/Home/projekte/ZIO-GIFGArchAM2014-ArcRefCloudv10p.pdf (besucht am 19.09.2014).
- [8] Olaf Prof. Dr. Zimmerman. "Architekturentscheidungen". Vorlesung Enterprise Computing HSR 2013/2014. Rapperswil, 2013.
- [9] Olaf Prof. Dr. Zimmerman. "Cloud Computing - Aus Sicht des Anwendungsarchitekten". OOP Software Meets Business 2014. München, 2014. URL: http://www.ifs.hsr.ch/fileadmin/user/_upload/customers/ifs.hsr.ch/Home/projekte/ARCCSichtAWA400Pv104p.pdf (besucht am 19.09.2014).
- [10] Olaf Prof. Dr. Zimmerman. "LAYERS UND TIERS, MIDDLEWARE, JAVA ENTERPRISE EDITION (JEE)". Vorlesung Application Architecture HSR 2014/2015. Rapperswil, Sep. 2014.
- [11] Jonathan Rasmusson. *The Agile Samurai*. P4.0. USA: Pragmatic Bookshelf, Aug. 2012. ISBN: 978-1-934356-58-6.
- [12] Marcel Tinner und Daniel Zigerlig. "Collaborative Decision management and Architectural refactoring (CDAR) Tool". Diss. Rapperswil: HSR Hochschule für Technik Rapperswil, 2014.
- [13] Wikipedia. *Cross-origin resource sharing - Wikipedia, the free encyclopedia*. 1. Dez. 2014. URL: http://en.wikipedia.org/wiki/Cross-origin_resource_sharing (besucht am 01.12.2014).
- [14] Wikipedia. *ISO/IEC 9126 – Wikipedia*. 26. Sep. 2014. URL: http://de.wikipedia.org/wiki/ISO/IEC_9126 (besucht am 26.09.2014).

- [15] Wikipedia. *Play framework - Wikipedia, the free encyclopedia*. 24. Sep. 2014. URL: http://en.wikipedia.org/w/index.php?title=Play_framework&oldid=625891581 (besucht am 24.09.2014).
- [16] Olaf Zimmermann. *IFS Olaf Zimmermann*. 14. Dez. 2014. URL: <http://www.ifs.hsr.ch/Olaf-Zimmermann.11623.0.html> (besucht am 14.12.2014).

Abbildungsverzeichnis

2.1	EEPPI bildet eine Brücke zwischen Entscheidungsmanagement- und Projektplanung	11
2.2	Metamapping in EEPPI: Verknüpfen von Entscheidungen und Aufgabenvorlagen	12
2.3	Übertragen: Ausführen von Prozessoren und Übermitteln der Aufgaben an ein Projektplanungstool	13
4.1	Laurin Murer	20
4.2	Tobias Blaser	20
4.3	Prof. Dr. Olaf Zimmermann	20
4.4	Atlassian Jira System Dashboard	21
5.1	Architektur Übersicht	22
5.2	Erweiterung CDAR / Integration	24
5.3	Architektur Varianten	26
5.4	Tier-Architektur	27
5.5	Servertechnologie-Vergleich: Prioritätsfindung	28
5.6	Servertechnologie-Vergleich: Vergleich der Technologien	28
5.7	Entwicklung von Webserver-Technologien der Top 10'000 Sites [6]	29
5.8	Ergebnis Servertechnologie-Vergleich	30
5.9	Servertechnologie	31
5.10	Clientseitige Programmiersprache	33
5.11	Clientseitiges Applikationsframework	34
5.12	Session State	36
5.13	Applikationsdatenfluss mit Beispieldatenquelle ADMentor	37
5.14	Konzeptionelle EEPPI-Domäne	39
5.15	ADRepo-Domäne	
	Bildlizenz(en): Eclipse Public License https://www.eclipse.org/legal/epl-v10.html Lukas Wegmann, IFS HSR https://www.ifs.hsr.ch	40
5.16	Metamapping	41
5.17	EEPPI-Implementationsdomäne	42
5.18	Strukturierungsmöglichkeiten von Tasks	44
5.19	Tasktemplates Strukturierung	45

5.20 Übertragen von Entscheidungen und Tasktemplates 47

5.21 Beispielprocessor für Issuetypes 48

5.22 Processorverwendung in einem Requesttemplate 48

5.23 Requesttemplate nach dem Ausführen des Processors 48

5.24 Verwendung von Variables in einem Requesttemplate 48

5.25 Übertragen von Tasks 49

5.26 Beispiel für die Verwendung einer Secondary Variable zum Verknüpfen des Eltern-Tasks 50

5.27 Mapping Methode 51

6.1 API-Dokumentation im Browser (Vollständige Dokumentation siehe Abschnitt E.1) 54

6.2 Annotations im DecisionKnowledgeSystemController 55

7.1 Wireframes Tobias 57

7.2 Wireframes Laurin 58

7.3 Dashboard & Tasks 58

7.4 Export Assistent 59

7.5 Export Assistent 59

7.6 EEPPI Home Screen im Browser 60

7.7 Problems & Tasktemplates 60

7.8 Administrationsbereich 61

7.9 Accountverwaltung 62

7.10 Übertragen von Tasks an ein Projektplanungstool 62

A.1 Symbolbild Persona Olivia Zander
Bildlizenz(en): CC BY 2.0 <https://creativecommons.org/licenses/by/2.0/> / Official GDC <https://www.flickr.com/photos/officialgdc/> . . 83

A.2 Symbolbild Persona Thomas Bucher
Bildlizenz(en): CC BY 2.0 <https://creativecommons.org/licenses/by/2.0/> / Steve wilson <https://www.flickr.com/photos/125303894@N06/> . 83

A.3 Übergeordnete User Stories (inklusive Entscheidungswissenssystem (DKS)) 85

B.1 Projektplan (Jira Versions/Meilensteine) 93

B.2 Beispiele von Jira Issues, sortiert nach Version 94

B.3 Jira Dashboard Beispiel 94

B.4 Git History in SmartGit 95

B.5 Github Releases mit Build-Archives 95

B.6 Angepassten Jira Issue-Workflow 96

B.7 Beispiel eines Unittests 96

B.8 Beispiel eines Integrationstests 97

B.9 Beispiel eines Behaviourtests (Selenium) 97

B.10 Trend der Testergebnisse auf dem Jenkins 98

B.11 Ergebnisse Client-Tests (Ansicht im Browser) 99

B.12 Total SLOC (Source Lines of Code) 100

B.13 Vergleich SLOC (Source Lines of Code) auf Server und Client inklusive Tests100

B.14 Testabdeckung auf dem Server und dem Client 101

B.15 Überblick Durchschnittsmesswerte	102
B.16 Anzahl Logical LOC pro Methode	102
B.17 Anzahl Parameter pro Methode	103
B.18 Cyclomatic Complexity pro Methode	103
B.19 Halstead Bugs pro Methode	104
D.1 Beispiel eines Request-Tempaltes	121
E.1 Erfassen von Arbeitszeit im Jira	166
E.2 Graph über die geleistete Arbeit	167
E.3 Graph über die geleistete Arbeit pro Woche	168
E.4 Risikomatrix	169
E.5 Bewertung Risiko 1	170
E.6 Bewertung Risiko 2	170
E.7 Bewertung Risiko 3	171
E.8 Bewertung Risiko 4	171
E.9 Bewertung Risiko 5	172

A. Anforderungen

Diese Kapitel beschreibt die Anforderungen an EEPPI in Form von User Stories und nicht-funktionalen Anforderungen.

A.1. Allgemeine Beschreibung

Softwarearchitekten¹ treffen und dokumentieren mit DKS²-Tools wie CDAR Architekturentscheidungen. EEPPI soll es ihnen ermöglichen, ein Mapping zwischen Architekturentscheidungen und Projekttasks zu erstellen. Ebenfalls soll es möglich sein, ein Mapping von Taskeigenschaften auf Felder einer API eines Projektplanungstools zu erstellen. Über diese beiden Mappings soll dem Architekten ermöglicht werden, aus Architekturentscheidungen Tasks in ein Projektplanungstool zu erstellen.

A.1.1. Benutzer-Charakteristik

Zielgruppe des EEPPI sind primär Softwarearchitekten und Projektplaner. EEPPI eignet sich jedoch grundsätzlich für Personen, die in einem Projekt tätig sind und ein Entscheidungswissenssystem in Kombination mit einem Projektplanungstool einsetzen wollen.

A.1.2. Einschränkungen

Voraussetzung für die Nutzung von EEPPI ist grundsätzliches Wissen über Softwarearchitektur, Architekturentscheidungen, Projektplanung sowie über die grundlegende Funktion eines Projektplanungstools. Also Wissen, wie es in Software Engineering Ausbildungen vermittelt wird, oder in Knowledge Hubs³ zu finden ist.

¹Einfachheitshalber wird jeweils auf die weibliche Form verzichtet, gemeint sind natürlich auch Softwarearchitektinnen. Dasselbe gilt auch für alle weiteren geschlechtsspezifischen Formulierungen.

²Entscheidungswissenssystem: Entscheidungswissensverwaltung, siehe auch Abschnitt A.2.2.

³Beispielsweise das Knowledge Hub des IFS Institut für Software der HSR: <http://www.ifs.hsr.ch/Architectural-Knowledge-Hubs.13193.0.html>

A.2. User Stories

A.2.1. Personas

Olivia Zander



Abbildung A.1.

52 Jahre alt. Olivia ist eine erfahrene **Softwarearchitektin** und ist seit 14 Jahren als solche tätig. Zurzeit arbeitet sie in einer kleinen Beratungsunternehmung, die Firmen beim Umstrukturieren von Softwareapplikationen unterstützt. Vor zwei Jahren absolvierte sie eine Weiterbildung im Bereich Cloud Computing. Seither hat sie selbst Erfahrungen mit Cloud Projekten sammeln können. Sie führte verschiedene Beratungsprojekten durch, deren Ziel es war, bestehende Anwendungen in die Cloud zu bringen.

Thomas Bucher



Abbildung A.2.

32 Jahre alt. Thomas hat vor acht Jahren in Rapperswil den Bachelor in Informatik abgeschlossen und arbeitet seither in der gleichen Firma wie Olivia. Er ist als **Projektplaner und technische Führungskraft** tätig und arbeitet je nach Projekt mit einem Team von 2-5 Entwicklern und Softwarearchitekten zusammen. Zurzeit unterstützt er eine externe Firma dabei, eine Anwendung mit täglich rund 10'000 Benutzern von ihren lokalen Servern in die Cloud zu transferieren.

A.2.2. Definitionen

Folgende Wörter werden in den User Stories verwendet und sind zum genauen Verständnis hier definiert.

Entscheidungswissensverwaltung (DKS: Entscheidungswissenssystem) System, das Entscheidungswissen, Entscheidungen, Optionen und getroffene Entscheidungen sowie deren Abhängigkeiten speichert. Beispiel: CDAR.

Wissensproduzent Person, die neues Wissen ins System (DKS und EEPPI) einpflegt (als Beispiel kann hier Olivia dienen).

Problem Space (Wissensbaum im CDAR) Ein Projekt in einem Entscheidungswissenssystem, in welchem ein Wissensproduzent Wissen ablegt.

Wissenskonsument Person, die einen Problem Space benutzt, um damit Entscheide zu fällen (als Beispiel kann hier Thomas dienen).

Solution Space (Entscheidungsprojekt im CDAR) Ein Projekt in einem Entscheidungswissenssystem, aus welchem ein Wissenskonsument Wissen konsumiert. Ein Solution Space wird aus einem Problem Space heraus durch Kopieren erzeugt.

Administrator Person, die für die Konfiguration und den Betrieb von EEPPI verantwortlich ist.

Abbildung Mit einer Abbildung lässt sich ein Datensatz in einen anderen Datensatz umwandeln.

erstellen aus Als Grundlage wird ein Datensatz genommen, aus welchem ein neuer Datensatz erstellt wird. Anschliessend sind die beiden Datensätze voneinander unabhängig und Änderungen an einem Datensatz beeinflussen den anderen Datensatz nicht.

Task Datensatz, welcher eine Aufgabe beschreibt.

Tasktemplate Datensatz, um später daraus Tasks in einem Projektplanungstool zu erstellen.

Entscheidungs-Task Task, welcher zum Treffen einer Entscheidung erledigt werden muss. Er ist einem Problem/Solution Space Item angehängt.

Operativer Task Task, welcher durch das Treffen einer Option entsteht. Er ist dementsprechend einer Option angehängt.

Problem Space Item Datensatz, der eine Wahl mit mehreren Optionen darstellt. Der Datensatz ist jedoch nur eine Vorlage, die Entscheidung kann noch nicht getroffen werden

Solution Space Item Datensatz, der eine Wahl mit mehreren Optionen darstellt. Er wird aus einer Entscheidungs-Vorlage erstellt. Die Entscheidung kann jetzt getroffen werden.

Entscheid Getroffenes (entschiedenes) Solution Space Item.

Option Möglichkeit, wie eine Entscheidung entschieden wird.

entscheiden Tätigkeit, in welcher für ein Solution Space Item der Entscheid gefällt wird.

importieren Anwenden einer Abbildung zur Aufnahme von Datensätzen in das EEPPI.

exportieren Anwenden einer Abbildung zur Ausgabe von Datensätzen aus dem EEPPI.

Projektplanungstool Externes Programm, welches Tasks verwaltet und für diese Tasks eine eigene Form erwartet.

API Schnittstelle eines Programms (sowohl bei externen, als auch bei EEPPI)

A.2.3. Übersicht über die User Stories

In der folgenden Abbildung A.3 werden die übergeordneten User Stories beschrieben, um den ganzen Prozess vom Entscheidungswissenssystem bis ins Projektplanungstool zu erfassen.

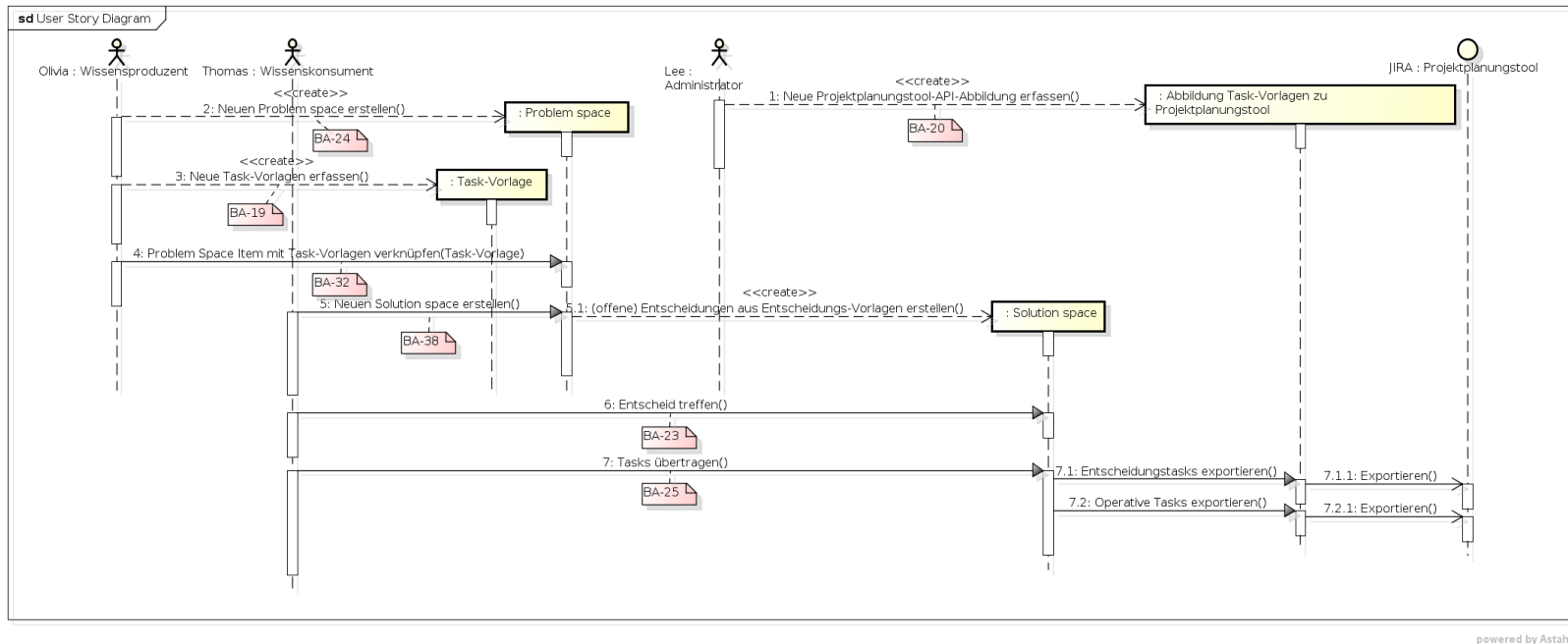


Abbildung A.3.: Übergeordnete User Stories (inklusive Entscheidungswissenssystem (DKS))

Dabei repräsentieren die drei Aktoren (Wissensproduzent, -konsument und Administrator) Personen wie in Abschnitt A.2.2 beschrieben, der Business-Aktor (ganz rechts) repräsentiert ein beliebiges Projektplanungstool und die roten Kästchen referenzieren den dazugehörigen Issue im EEPPI-Projektplanungstool. Nachfolgend sind die Erklärungen für die aufgezeigten User Stories. Die User Stories sind jeweils im Format nach Mike Cohn [11] beschrieben:

As a <type of user>,
I want <to perform some task>
so that I can <achieve some goal/benefit/value>.

Core user stories (BA JIRA)

JQL Query: project = BA AND issuetype = Story AND key IN ("BA-19", "BA-20", "BA-23", "BA-24", "BA-25", "BA-32", "BA-38") ORDER BY key ASC
Sorted by: Key ascending

1–7 of 7 as at: **13.12.14 21:10**

Key	Summary	Description	Scope
BA-19	neue Task-Vorlagen erfassen	Als Wissensproduzent, will ich Task-Vorlagen erstellen um sie mit Problem Space Items verknüpfen zu können.	Current project (EEPPI)
BA-20	neue Projektplanungstool-API-Abbildung erfassen	Als Administrator will ich verschiedene Abbildungen von Task-Vorlagen auf Projektplanungstool-APIs erstellen um damit Benutzern den Export von Tasks in ein Projektplanungstool ermöglichen zu können, damit Benutzer verschiedene Projektplanungstools benutzen können.	Current project (EEPPI)
BA-23	Entscheid treffen	Als Wissenskonsument will ich Entscheide für Solution Space Items treffen um damit weitere Solution Space Items zu sehen und weitere Entscheide treffen zu können.	Existing project (DKS)
BA-24	neuen Problem Space erstellen	Als Wissensproduzent will ich Problem Space Items erfassen und untereinander verknüpfen um mein Wissen an Wissenskonsumenten weiter zu geben, und damit dieses bestehende Wissen nutzen und damit effizienter arbeiten können.	Existing project (DKS)
BA-25	Tasks übertragen	Als Wissenskonsument will ich Tasks (Operative Tasks und/oder Entscheidungstasks) in ein Projektplanungstool exportieren um sie darin verwalten zu können.	Current project (EEPPI)

Key	Summary	Description	Scope
BA-32	Problem Space Item mit Task-Vorlagen verknüpfen	Als Wissensproduzent will ich einem Problem Space Item Task-Vorlagen zuordnen können um die Abbildung von Entscheidungs-Vorlagen auf Task-Vorlagen zu realisieren.	Current project (EPPPI)
BA-38	neuen Solution Space erstellen	Als Wissenskonsument will ich aus einem Problem Space einen neuen Solution Space erstellen um damit für ein konkretes Projekt Entscheidungen zu treffen, damit ich Entscheide nachvollziehbar dokumentieren kann.	Existing project (DKS)

Advanced user stories (BA JIRA)

JQL Query: project = BA AND issuetype = Story AND key not in (BA-19, BA-20, BA-23, BA-24, BA-25, BA-32, BA-38) ORDER BY key ASC

Sorted by: Key ascending

1–6 of 6 as at: 13.12.14 21:13

Key	Summary	Description	Scope
BA-29	Taskeigenschaften erstellen	Als Administrator will ich Taskeigenschaften erstellen um Benutzer das Erfassen von Eigenschaften wie z.B. "Erwarteter Aufwand", "Beschreibung" oder "Typ" auf Task-Vorlagen zu ermöglichen.	Current project (EEPPI)
BA-37	Neue Abbildung DKS => EEPPI erfassen	Als Administrator will ich verschiedene Datenquellen für Solution und Problem Spaces konfigurieren um damit Benutzern das Auflisten von Solution und Problem Spaces zu ermöglichen, damit Benutzer Entscheidungen von verschiedene Tools verwenden können.	Current project (EEPPI)
BA-39	DKS Space mit PPT Projekt verknüpfen	Als Wissenskonsument will ich Solution Spaces (eines DKS) mit Projekten des Projektplanungstools verknüpfen um Tasks in das richtige Projekt übertragen zu können, damit ich Zeit spare beim Projektmanagement (Automatisierung) und weniger Tasks vergesse (Qualitätssteigerung, weniger Fehler).	Current project (EEPPI)
BA-40	Zugriffsrechte und Fähigkeiten	Als Administrator will ich Rollen und Zugriffsrechte verwalten können um Benutzern auf die Auswahl an Solution Space Items ihrer Soluton Spaces (im DKS) und die Projektauswahl (im Projektplanungstool) einschränken zu können damit sie effizienter Arbeiten können, weniger Fehler machen und ihre Entscheidungen & Tasks vor der Einsicht und der Manipulation anderer	Current project (EEPPI)

Key	Summary	Description	Scope
		geschützt sind.	
BA-41	Report Task Übertragung	Als Wissenskonsument will ich sehen können, welche Tasks wann ins Projektplanungstool übertragen wurden, um mir eine Übersicht über bereits übertragene Tasks verschaffen zu können, damit ich weniger Zeit brauche beim Übertrag von zukünftigen Tasks.	Future project
BA-61	Task-Vorlagen Strukturierung	Als Wissenproduzent will ich Task-Vorlagen strukturieren und einfach wieder auffinden können um wenig Zeit zum Verknüpfen von Task-Vorlagen und Problem Space Items zu benötigen und doppelte Task-Vorlagen zu vermeiden.	Future project

A.3. Weitere Anforderungen

A.3.1. Qualitätsmerkmale

Die folgenden Qualitätsmerkmale orientieren sich an ISO/IEC 9126 [14] und beziehen sich, wenn nichts anderes angegeben ist, auf alle User Stories.

A.3.1.1. Functionality

Interoperabilität Die *Schnittstelle der Datenquelle* der Entscheidungen sowie die *Schnittstellen für das Projektplanungstool* sollen konfigurierbar gestaltet sein, sodass sie für andere Systeme als CDAR/ADRepo und Jira/Redmine umkonfiguriert werden können. Für *Software eigene Schnittstellen* sollen Aufrufparameter, Rückgabewerte und jeweils mindestens zwei Beispielaufrufe und Beispielantwortdatensätze dokumentiert werden, sodass zukünftige Projekte mit EEPPI interagieren können.

Sicherheit Im System gespeicherte Daten sollen vor Zugriff Dritter geschützt werden. Es soll nicht möglich sein, Informationen über Entscheidungen, Tasks oder Benutzer auszulesen oder zu verändern, ohne authentisiert zu sein.

Konformität Zur Umsetzung der Schnittstellen und des Mapping Konzeptes sollen die folgenden existierende Formate und Protokolle eingesetzt werden: HTTP, JSON, REST.

Kompatibilität Die Clientapplikation soll mit modernen Browsern (Firefox/Chrome/Internet Explorer/Safari, jeweils neuste zwei Versionen) kompatibel sein. Ältere Versionen sollen nicht unterstützt werden.

A.3.1.2. Zuverlässigkeit

Robustheit Das System soll auch im Fehlerfall einen konsistenten Zustand annehmen und den Benutzer in Form einer lesbaren Fehlermeldung über das Problem informieren, sowie Anleitung zum Beheben des Fehlers geben.

A.3.1.3. Usability

Verständlichkeit Ein Benutzer soll anhand der Dokumentation selbstständig die Mapping Konzepte erlernen und erstellen können, um damit Tasks generieren zu können. Das selbstständige Einarbeiten soll nicht länger als einen halben Arbeitstag dauern.

Tasktemplate Mapping Die Darstellung des Tasktemplate Mappings soll mit bis zu 50 gemappten Tasktemplates gleich flüssig und angenehm bedienbar sein wie mit 5 gemappten Tasktemplates.

Tasktemplate Listing Die Darstellung der Tasktemplates wird im Durchschnitt 10-30 Einträge enthalten und soll mit bis zu 100 Einträgen flüssig bedienbar sein.

A.3.1.4. Portability

Anpassbarkeit Die Software soll keine plattformspezifische Konfiguration oder Modifikation an der Umgebung (Betriebssystem, Virtuellen Maschine, Webserver) erfordern und soll somit mindestens bei Heroku⁴ installiert werden können.

Installierbarkeit Die Installation von EEPPI soll gut dokumentiert sein, sodass die Installation und Konfiguration der Anwendung nicht mehr als drei Stunden dauert.

A.4. Sicherheit

Eine Applikation wie EEPPI, die geschäftsrelevante Daten beinhaltet, muss entsprechend gegen Zugriffe Dritter abgesichert werden.

Am 6. Oktober 2014 wurde beim Meeting mit dem Betreuer als Ansprechpartner der Kundengruppe entschieden, dass Mandantenfähigkeit eine untergeordnete Rolle spielt. Entsprechend wurde entschieden, dass Sicherheitsanforderungen geringerer Priorität sind als funktionelle Anforderungen. EEPPI ist ein Forschungsprojekt und dient als Grundlage für weitere Forschung und Entwicklung. Es ist nicht das Ziel, EEPPI nach Ende der Bachelorarbeit direkt öffentlich zugänglich über das Internet zur Verfügung zu stellen, sondern als Forschungssystem in einem internen Netz zu betreiben.

Trotzdem sollen in EEPPI einige grundlegende Sicherheits-Funktionen umgesetzt und deren Erweiterungsmöglichkeiten berücksichtigt werden. Session- und Passwortmanagement sollen gemäss heute üblichen Richtlinien umgesetzt werden. Mögliche Erweiterungen, beispielsweise eine OAuth-Anmeldung bei Projektplanungstools oder Rechte- und Rollenkonzepte, sollen beim Design berücksichtigt und als mögliche Erweiterungen (Abschnitt 9.6) dokumentiert werden.

⁴Heroku Cloud applications: <https://www.heroku.com/>

B. Projekt- und Qualitätsmanagement

Zur Verwaltung des Projektes hat das Projektteam ein Jira eingesetzt.

B.1. Projektmanagement

Zur Grobplanung und zur Planung der Meilensteine wurden Jira-Versionen eingesetzt. Die Meilensteine hat das Projektteam zu Beginn des Projekts definiert und zeitlich festgelegt. Diese Planung ist auch in Abbildung B.1 zu sehen.

Für die Architektur-Meilensteine und die Meilensteine zum Abschluss der Dokumentation wurde bereits bei der Grobplanung deren Inhalt entschieden. Für die Entwicklungsmeilensteine wurde, wie im agilen Projektmanagement üblich, deren Inhalt jeweils zu Beginn zusammen mit dem Betreuer als Ansprechpartner der Kundengruppe definiert.











Name	Start date	Release date	Description
 Release BA	08.09.14	16.12.14	Documentation reviewed, ready for delivery, poster finished, abstract completed
 Doc.Completion	24.11.14	13.12.14	Documentation completed, ready for review
 Dev.Completion	24.11.14	04.12.14	Code refactored and frozen
 Dev.Milestone3	10.11.14	26.11.14	Development completed (Feature freeze)
 Dev.Milestone2	27.10.14	10.11.14	Development milestone 1, topic tbd
 Dev.Milestone1	13.10.14	27.10.14	Development milestone 1: login, DKS adapter, wireframes
 Dev.Prototype	29.09.14	13.10.14	Prototype completed to fix high risk issues
 Arch.ArchitectureDetail	29.09.14	06.10.14	Analysis done, technology fixed, architecture concept elaborated, risks known
 Arch.ArchitectureConcept	22.09.14	29.09.14	Analysis draft, technology draft, deployment draft, architecture concept draft, risks management draft ready to review
 Infrastruktur & Admin	15.09.14	22.09.14	Infrastructure ready, administrative tasks done

Abbildung B.1.: Projektplan (Jira Versions/Meilensteine)

Features haben wir in Zusammenarbeit mit dem Vertreter der Kundengruppe priorisiert und daraus resultierende Issues Meilensteine zugeordnet. Zur Strukturierung haben wir zusätzlich Labels eingesetzt.

Key	Summary	Labels	Scope	Assignee	Due	Fix Versions	P
BA-172	Improve api	None		Laurin Murer	24.11.14	Dev Completion	↑
BA-203	Play crashes on saving taskTemplates with parent reference	None		Laurin Murer	01.12.14	Dev Completion	↑
BA-158	Refactor server side code	ServerSide		Laurin Murer	01.12.14	Dev Completion	↑
BA-195	Transmissioncontroller fails on rendering TaskTemplates	None		Laurin Murer	01.12.14	Dev Completion	↑
BA-117	Move everything from public to assets directory	ServerSide		Unassigned		Dev Completion	↑
BA-196	Create full workflow tests	None		Unassigned		Dev Completion	↓
BA-193	Create example Request Template for Redmine	None		Tobias Blaser	01.12.14	Dev Completion	↓
BA-174	Security is no requirement -> document	None		Unassigned	08.12.14	Doc. Completion	↑
BA-194	Document how we manage the project (Jira, communication, reviews, ...)	None		Tobias Blaser	01.12.14	Doc. Completion	↑
BA-204	Difference between task template & request template documented?	None		Tobias Blaser	08.12.14	Doc. Completion	↑
BA-161	Document future work possibilities	None		Unassigned	08.12.14	Doc. Completion	↑
BA-160	Document differences between wireframes and reality	None		Unassigned	08.12.14	Doc. Completion	↑
BA-205	Create simple tutorial for EEPPI, include in doku and EEPPI	None		Tobias Blaser	08.12.14	Doc. Completion	↑
BA-176	Add list with used components and licenses to documentation	None		Unassigned	08.12.14	Doc. Completion	↑
BA-177	Document mapping concept overview (metamapping)	None		Unassigned	08.12.14	Doc. Completion	↑
BA-175	Document future features	None		Unassigned	08.12.14	Doc. Completion	↑
BA-206	Update stories (remove cdar dependencies)	None		Unassigned	08.12.14	Doc. Completion	↑
BA-173	Document reason for writing no information back to disk	None		Unassigned	08.12.14	Doc. Completion	↑
BA-178	Is "Export is a one-way procedure" and why documented?	None		Unassigned	08.12.14	Doc. Completion	↑
BA-190	Document mapping of DKS elements to EEPPI elements	None		Unassigned	01.12.14	Doc. Completion	↑

Abbildung B.2.: Beispiele von Jira Issues, sortiert nach Version

Anhand den geschätzten Aufwänden pro Issue und der zur Verfügung stehenden Zeit eines Meilensteine haben wir jeweils eine Meilensteinplanung durchgeführt. Dabei haben wir maximal 2/3 der zur Verfügung stehenden Zeit für Issues eingeplant und den Rest für Unvorhergesehenes, Meetings und Planung vorgesehen.

System Dashboard

Filter Results: My Work in Progress

T	Key	P	Summary	Due
🔍	BA-157	↑	Refactor client side code	01.12.14

1-1 of 1

Filter Results: My Issues to Review

T	Key	P	Summary	Due
🔍	BA-200	↑	Prevent creation of data on API-Documentation call	
🔍	BA-199	↑	BA-158 / Apply code check of idea and make refactoring based on findings	
🔍	BA-187	↑	Replace paths.ts by paths.js and then path.js by paths.scala.js	

1-3 of 3

Filter Results: My Further Assigned Issues

T	Key	P	Summary	Fix Versions	Due
🔍	BA-193	↓	Create example Request Template for Redmine	Dev Completion	01.12.14
🔍	BA-194	↑	Document how we manage the project (Jira, communication, reviews, ...)	Doc. Completion	01.12.14
🔍	BA-204	↑	Difference between task template & request template documented?	Doc. Completion	08.12.14
🔍	BA-205	↑	Create simple tutorial for EEPPI, include in doku and EEPPI	Doc. Completion	08.12.14
🔍	BA-108	↓	Inform user about network/dks/ppt connection		

1-5 of 5

Filter Results: Unassigned Issues (this version)

T	Key	P	Summary	Due
🔍	BA-196	↓	Create full workflow tests	
🔍	BA-117	↑	Move everything from public to assets directory	

1-2 of 2

Activity Stream

Your Company JIRA

Today

Tobias Blaser changed the status to Ready to Review on BA-174 - Check if server redirect is documented with a resolution of Fixed

- Logged '30 minutes'
- Changed the Remaining Estimate to '30 minutes'

30 minutes ago Comment Watch

Tobias Blaser updated 2 fields of BA-174 - Check if server redirect is documented

- Logged '30 minutes'
- Changed the Remaining Estimate to '30 minutes'

25 minutes ago Comment Watch

Laurin Murer changed the status to Ready to Review on BA-187 - Replace paths.ts by paths.js and then path.js by paths.scala.js with a resolution of Fixed

- Logged '1 hour, 15 minutes'
- Changed the Remaining Estimate to '2 hours'

31 minutes ago Comment Watch

Laurin Murer updated 2 fields of BA-187 - Replace paths.ts by paths.js and then path.js by paths.scala.js

- Logged '1 hour, 15 minutes'
- Changed the Remaining Estimate to '2 hours'

31 minutes ago Comment Watch

Abbildung B.3.: Jira Dashboard Beispiel

Jira bietet anpassbare Dashboards, die einen Überblick über das laufende Projekt bieten.

Der Activity Stream von Jira sowie die Git History ermöglichten es uns auf einfache Weise nachzuvollziehen, an was der Teampartner in den letzten Stunden gearbeitet hat. Dies senkt den Kommunikationsaufwand und die Notwendigkeit, jederzeit gemeinsam an einem Ort zu arbeiten.

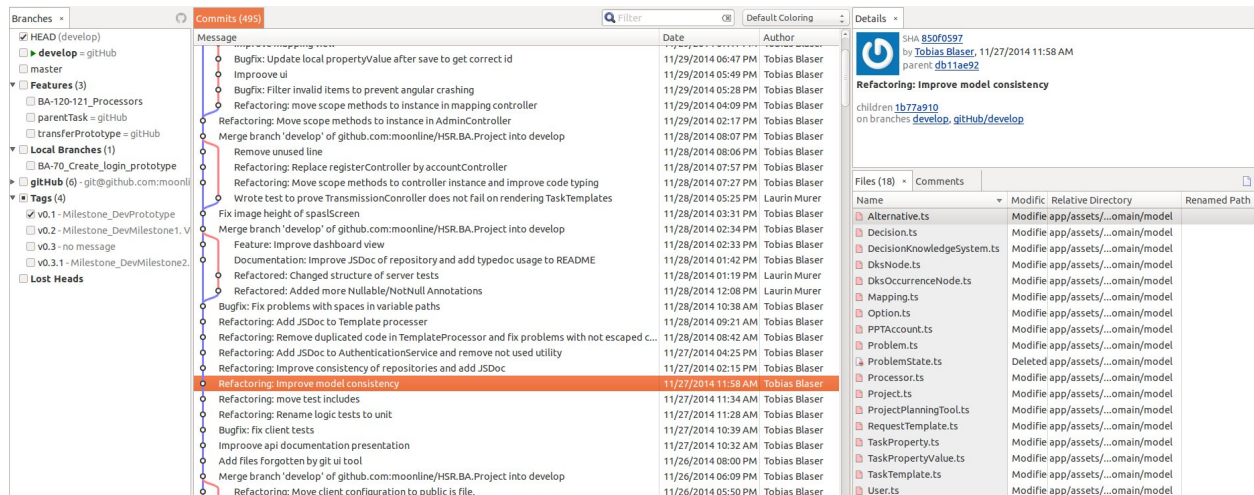


Abbildung B.4.: Git History in SmartGit

Für grössere Features haben wir Git Flow Featurebranches eingesetzt. Für Releases entsprechend Releasebranches. Zusätzliche haben wir die Funktion «Releases» von Github zum Hinzufügen von fertigen Builds zu Releases genutzt.

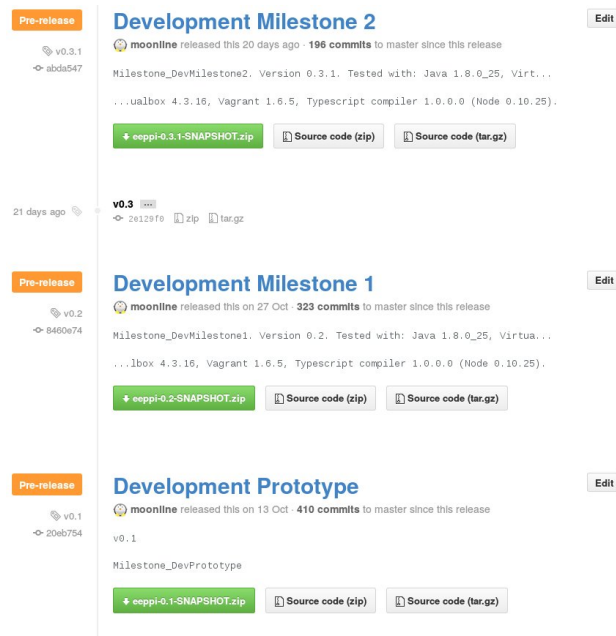


Abbildung B.5.: Github Releases mit Build-Archives

B.2. Qualitätssicherung

Um sicherzustellen, dass keine Issues geschlossen werden, ohne dass die Arbeit einem Review unterzogen wurde, haben wir den Issue Workflow im Jira entsprechend gestaltet.

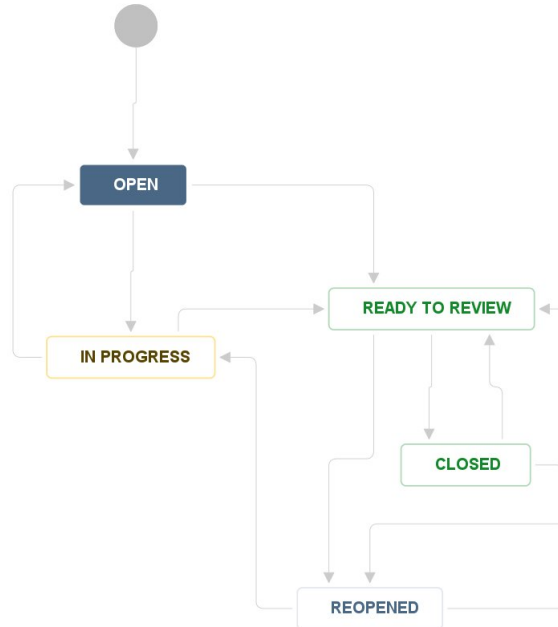


Abbildung B.6.: Angepassten Jira Issue-Workflow

Fertig gestellte Issues müssen immer dem andern Teammitglied zum Review gesandt werden und tauchen entsprechend auf dessen Dashboard als «Ready to Review» auf.

Es geht dabei nicht darum, für jeden erledigten Issue den kompletten Code des Andern anzusehen, sondern das Resultat grob anzuschauen und eventuell Edge-Cases¹ zu prüfen. Ein komplettes Code Review jedes Issues wäre zeitlich nicht verhältnismässig.

B.3. Testkonzept

Um zuverlässig alle notwendigen Bereiche mit Tests abzudecken, werden folgende Tests durchgeführt:

Unit (Unit- /Logiktests) Tests, die eine einzelne Klasse, Service oder Komponente testen. Andere Klassen sind nur soweit Teil des Tests, wie dies aufgrund der Abhängigkeiten notwendig ist.

```

@Test
public void createUserTest() {
    AbstractTestDataCreator.createUser("Hans", "1234");
    User user = new UserDAO().readByName("Hans");
    assertThat(user.getName()).isEqualTo("Hans");
}
    
```

Abbildung B.7.: Beispiel eines Unittests

¹Spezialfälle, bezogen auf Input Daten oder Workflows der grafischen Oberfläche

Integration (Integrationstests) Tests, die das Zusammenspiel zwischen Klassen, Komponenten und Services testen. Diese Tests können über mehrere Layers bis mehrere Tiers laufen. Alle API-Aufrufe werden mit solchen Tests getestet.

```
@Test
public void testLoginSuccessful() throws Throwable {
    //Setup
    User user = AbstractTestDataCreator.createUserWithTransaction("Hansli", "1234");
    //Test
    Result result = callPostAction(controllers.user.routes.ref.UserController.login(), postData(...));
    //Verification
    assertThat(status(result)).isEqualTo(OK);
    assertCheckJsonResponse(result, Json.toJson(user));
    verifyLoggedIn(user, result, true);
}
```

Abbildung B.8.: Beispiel eines Integrationstests

Behaviour (Verhaltenstests) Gross-Integrationstests, die über alle Tiers und Layers laufen. Diese Tests testen Workflows von der Persistenz bis zur grafischen Ausgabe im Userinterface. Dazu wird mit Selenium ein Browser gestartet und dessen Ausgabe analysiert. In EPPPI sind alle Haupt-Userstories mit einem Selenium-Test abgedeckt.

```
@Test
public void registerAndLoginTest() {
    String username = "New User";
    String password = "1234";
    browser.goTo("/");
    browser.click("a", withText("Account"));
    browser.await().untilPage().isLoaded();
    browser.fill("#registerUserName").with(username);
    browser.fill("#registerPassword").with(password);
    browser.fill("#registerPasswordRepeat").with(password);
    browser.click("button", withText().contains("Register"));
    await(() -> browser.find(".messagebox.success", withText("Registration successful.")));
    doLogin(username, password);
    await(() -> browser.find("a", withText("Problems & Task Templates")));
    await(() -> browser.find("a", withText("Administration")));
    browser.click("button", withText("Logout"));
    assertThat(browser.find("a", withText("Problems & Task Templates")).isEmpty());
    assertThat(browser.find("a", withText("Administration")).isEmpty());
}
```

Abbildung B.9.: Beispiel eines Behaviourtests (Selenium)

Um die Schnittstellen zusammen mit den echten Systemen zu testen, werden die externen Systeme mit einer Vagrant-Umgebung (siehe auch Abschnitt C.3.4) gestartet. Konkret läuft ein Test nach folgendem Muster ab:

1. Erstellen und starten aller benötigten Vagrant Systeme
2. Durchführen der Tests
3. Löschen der gestarteten Vagrant Systeme

B.4. Testergebnisse

Der von uns eingerichtete Buildserver Jenkins (siehe Abschnitt C.2.9.1) bezog automatisch neue Commits aus dem Repository und führte nach jeder Änderung einen Testdurchlauf aus.

Die folgende Grafik zeigt die Testdurchläufe zwischen dem 1. und 4. Dezember. Gelbe Testresultate stehen für nicht ausgeführte Tests. In unserem Falle betrifft dies alle Tests, die Vagrant erfordern, da die Leistung auf dem Testserver nicht zur Ausführung dieser reichte. Rote Testresultate bedeuten fehlgeschlagene Tests, grüne erfolgreiche.



Abbildung B.10.: Trend der Testergebnisse auf dem Jenkins

Einer der dabei laufenden Tests ist ein Selenium-Test, der die Client-Tests über einen Browser ausführt und dessen Ausgabe analysiert. Ein erfolgreicher Durchlauf dieser wird in Abbildung B.11 gezeigt. Total besitzt EEPPI rund 150 Tests, davon testen rund 1/3 den Clientteil, knapp 2/3 den Serverteil und rund 10 Tests übergeordnete Funktionalitäten sowie das Zusammenspiel zwischen Client und Server.

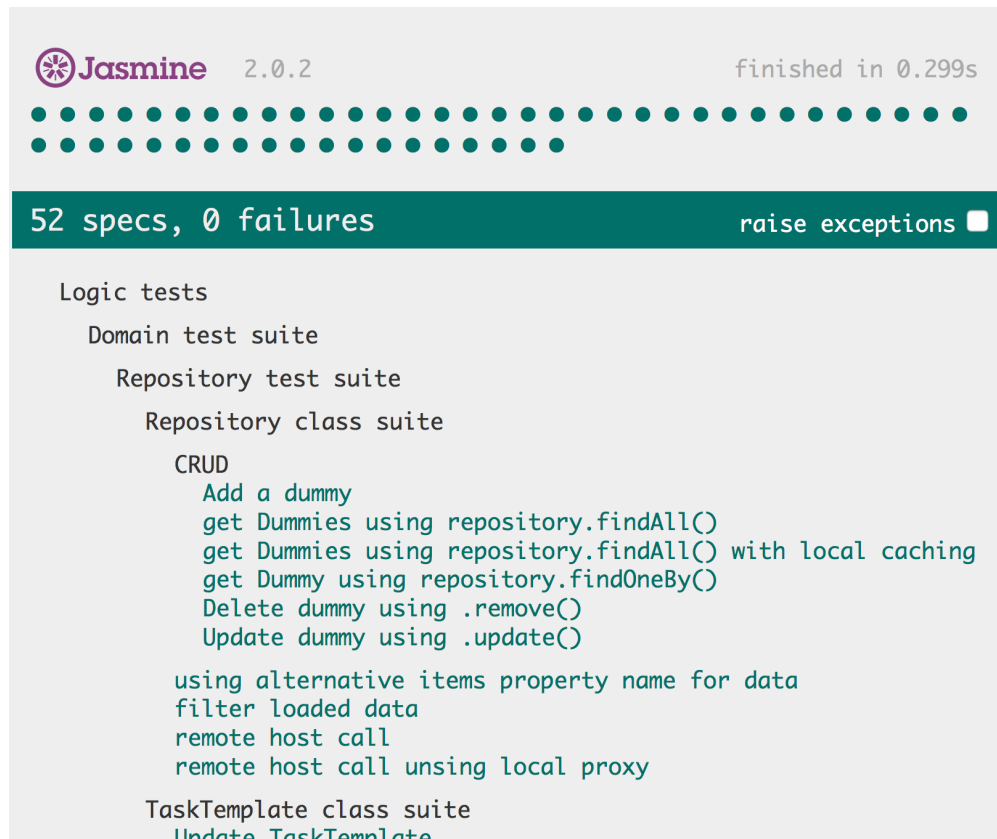


Abbildung B.11.: Ergebnisse Client-Tests (Ansicht im Browser)

B.5. Metriken

Zur Analyse von EEPPI haben wir drei verschiedene Arten von Metriken erstellt. Die erste zeigt den Umfang des Projekts, die zweite wie gut wir getestet haben und die dritte gibt Auskunft über die Code Qualität.

Auf dem Client wurde mit TypeScript entwickelt, dieses wurde zu JavaScript kompiliert. Die Metriken über den Code Umfang wurden für den Clientteil mit den TypeScript-Dateien durchgeführt, die restlichen Metriken aufgrund der mangelnden Verfügbarkeit an Tools für TypeScript direkt mit dem JavaScript. Dies hat soweit einen Einfluss, dass die Qualität des von TypeScript erzeugten Codes die Messung beeinflusst.

B.5.1. Umfang

Total setzt sich EEPPI aus gut 25'000 Zeilen Code zusammen, inklusive Leerzeilen, Kommentaren, Konfigurationsdaten und Code von anderen Entwicklern (siehe Kapitel 8.2). Die Zusammensetzung ist in Abbildung B.12 zu sehen.

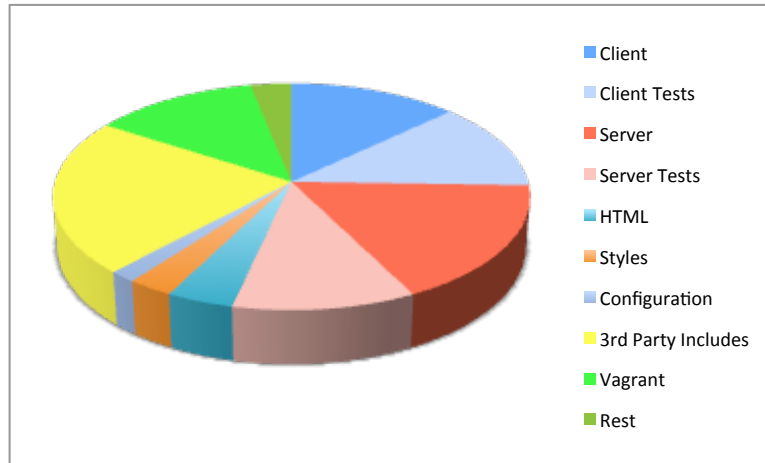


Abbildung B.12.: Total SLOC (Source Lines of Code)

Gut die Hälfte des Umfangs entfällt dabei auf unsern Client- und Servercode und seine Tests. Ein weiterer grosser Anteil stellen die externen Libraries und Frameworks (22%), sowie die Daten für die Initialkonfiguration der Vagrant-Umgebungen. Die restlichen 12% des Projektes setzen sich zusammen aus Konfiguration, Templates für die Benutzeroberfläche und Styles.

Wird der Hauptteil von EEPPI betrachtet (effektiver ausführbarer Code Server- und Client), ist zu sehen, dass die Server- und Client in der gleichen Grössendimension liegen. Dies war zu erwarten, da sie auch den gleichen Umfang der Daten verarbeiten.

Auch die Tests liegen in der gleichen Grössenordnung. Allerdings sind im Client Test Code viele Zeilen an Initialisierungsdaten enthalten zum Mocken² der Schnittstellenauf-rufe. Die effektive Grösse der Client Tests liegt ca. 25% tiefer.

Die detaillierte Aufteilung und genaue Anzahl der SLOC (Source Lines of Code) der eigenen Codeteile ist in Abbildung B.13 zu sehen.

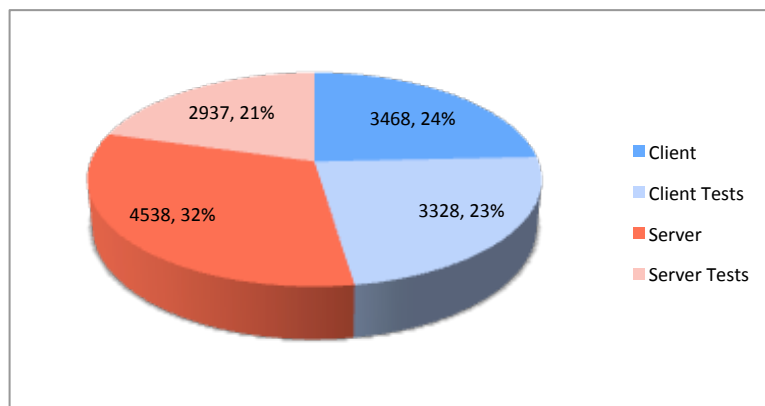


Abbildung B.13.: Vergleich SLOC (Source Lines of Code) auf Server und Client inklusive Tests

²Das verwendete Test-Framework Angular Jasmine erlaubt das simulieren (mocken) der REST-Schnittstelle

B.5.2. Test Coverage

Sowohl auf dem Server als auch auf dem Client sind wir mehrheitlich nach TDD³ vorgegangen, dementsprechend hoch ist die Testabdeckung. Ziel war von Anfang an nicht 100% Testabdeckung zu erreichen, sondern die wichtigsten Funktionen zu testen.

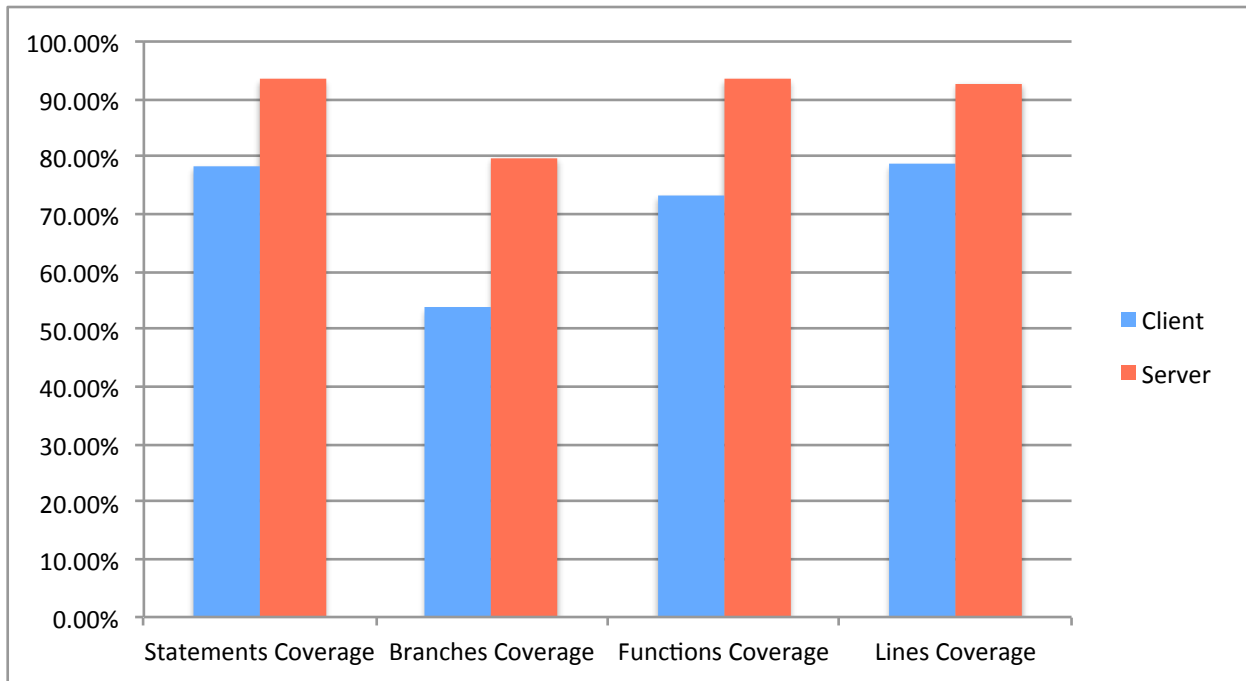


Abbildung B.14.: Testabdeckung auf dem Server und dem Client

Abbildung B.14 zeigt die Testabdeckung auf. Es ist zu sehen, dass auf dem Server die Testabdeckung höher ist als auf dem Client, aber auch auf dem Client sind die zentralen und wichtigen Elemente gut abgedeckt.

B.5.3. Qualität

Die Codequalität ohne grosse Verfälschung zu messen ist eine grosse Herausforderung. Hier zeigt sich der erwähnte Einfluss des generierten JavaScripts besonders stark. TypeScript generiert viele anonyme, direkt ausgeführte Funktionen, die der Kapselung des Codes dienen. Der so erzeugte Code drückt die Metriken in die Höhe. Trotzdem lassen sich einige Aussagen treffen anhand der gemessenen Daten.

Wir haben die wichtigsten Kennzahlen evaluiert, so die Anzahl der Logical LOC (Anzahl effektiver Code-Zeilen) pro Methode, die Anzahl der Parameter und die Cyclomatic Complexity⁴.

³Test Driven Development

⁴Komplexitätswert der die Operationen pro Methode repräsentiert <http://www.aivosto.com/project/help/pm-complexity.html>

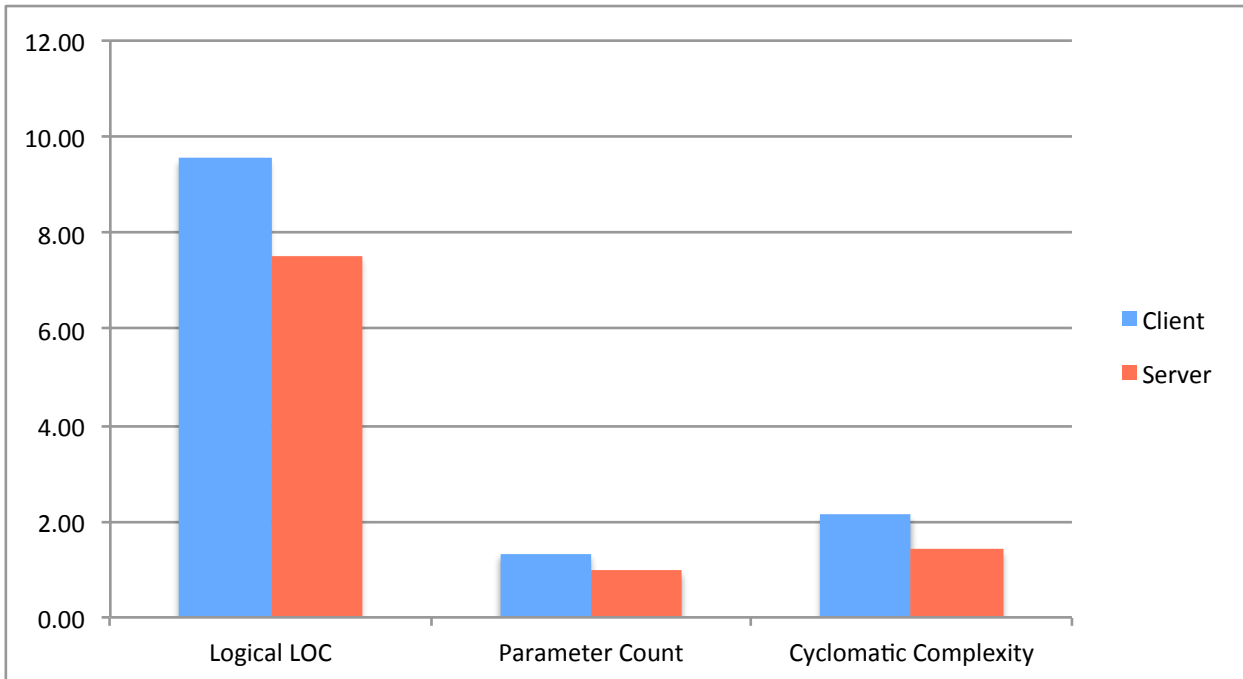


Abbildung B.15.: Überblick Durchschnittsmesswerte

In Abbildung B.16 ist die Anzahl der logischen Zeilen Code pro Methode ausgewiesen. Gut sichtbar ist, dass auf dem Server wie auf dem Client rund 35% aller Methoden drei logische Codezeilen aufweisen. Dies sind primär Getter und Setter, bzw. auf dem Client anonyme, von TypeScript erzeugte Methoden.

Zusammengefasst kann man sagen, dass die Mehrheit der Methoden mit weniger als zehn Zeilen Code auskommen und nur einige wenige Methoden mehr Zeilen aufweisen.

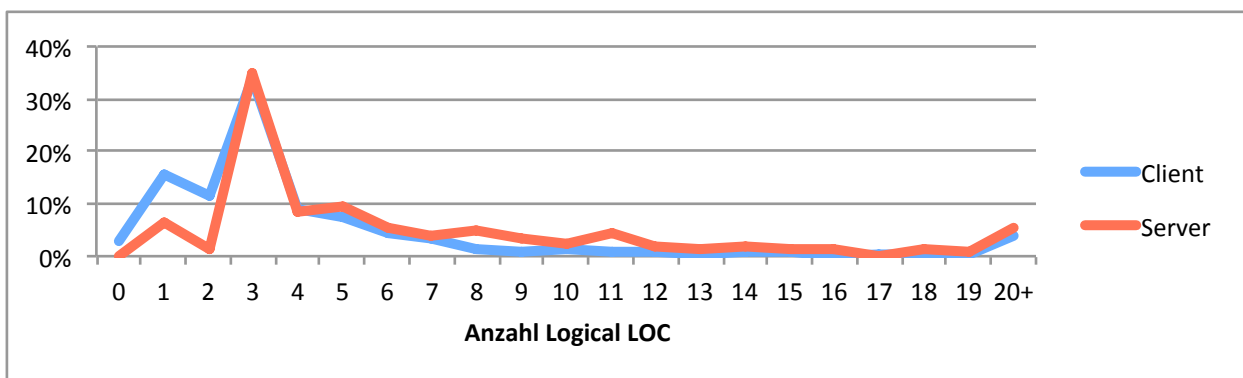


Abbildung B.16.: Anzahl Logical LOC pro Methode

In Abbildung B.17 wird die Häufigkeit der Anzahl Methoden Parameter aufgezeigt. Gut erkennbar ist die Tatsache, dass sowohl auf dem Server, wie auch auf dem Client, rund 3/4 der Methoden keinen oder nur einen einzigen Parameter besitzen. Die Anzahl Methoden mit keinem Parameter ist auf dem Client signifikant tiefer, da aufgrund der Properties in JavaScript das implementieren von Getter für Public-Attributes entfällt.

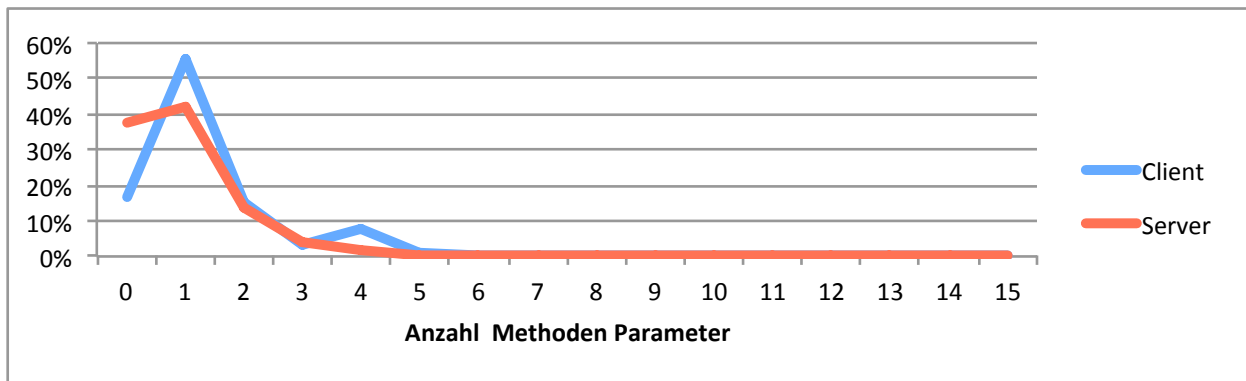


Abbildung B.17.: Anzahl Parameter pro Methode

Abschliessend ist die Cyclomatic Complexity in Abbildung B.18 aufgezeichnet. Sie beschreibt, wie viele Operationen eine Methode enthält. Die Spitzen bei Eins repräsentieren auch hier die Getter und Setter.

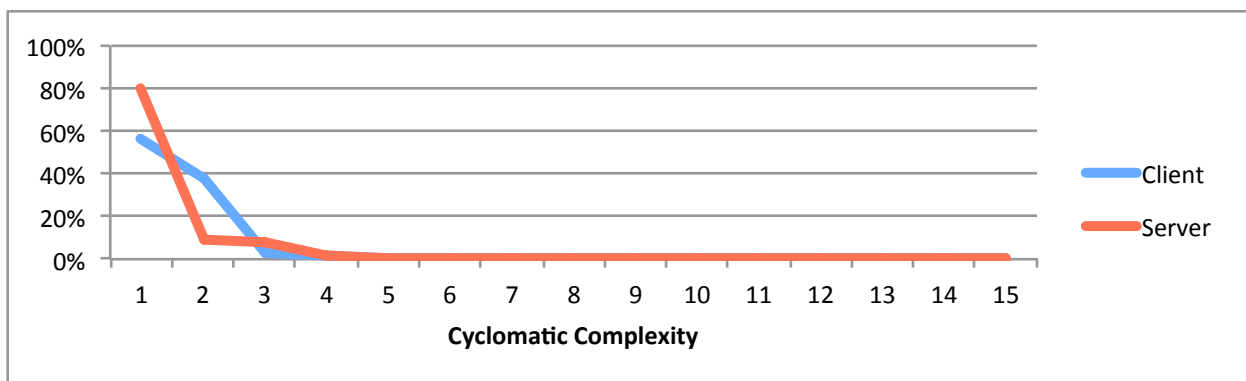


Abbildung B.18.: Cyclomatic Complexity pro Methode

B.5.4. Fazit

Wir sind mit dem erreichten Umfang sowie mit der Testabdeckung von EEPPI sehr zufrieden. Die effektive Höhe der Codequalität lässt sich jedoch nur schwer eruieren.

Ungefähr in der Mitte des Projekts hat ein Mitarbeiter des IFS⁵ einen Codereview durchgeführt und positive Bilanz gezogen. Zusätzlich sind aufgrund dieses Feedbacks Verbesserungen in den Code eingeflossen.

Maurice Halstead hat eine Softwaremetrik entworfen, welche die Komplexität der Software berechnet und davon ausgehend einige Kennzahlen liefert. Eine davon ist die «Halstead Bugs», sie gibt die Fehlerkomplexität an, also wie viele Fehler die Software aufgrund der berechneten Komplexität und Grösse statistisch ungefähr enthalten könnte. Das heisst aber nicht, dass die Software so viele Fehler enthält, sondern die Abstraktion mit Bugs ist nur für das bessere Verständnis der Grössenordnung gedacht.

⁵Institut für Software, HSR Hochschule für Technik Rapperswil, <http://www.ifs.hsr.ch>

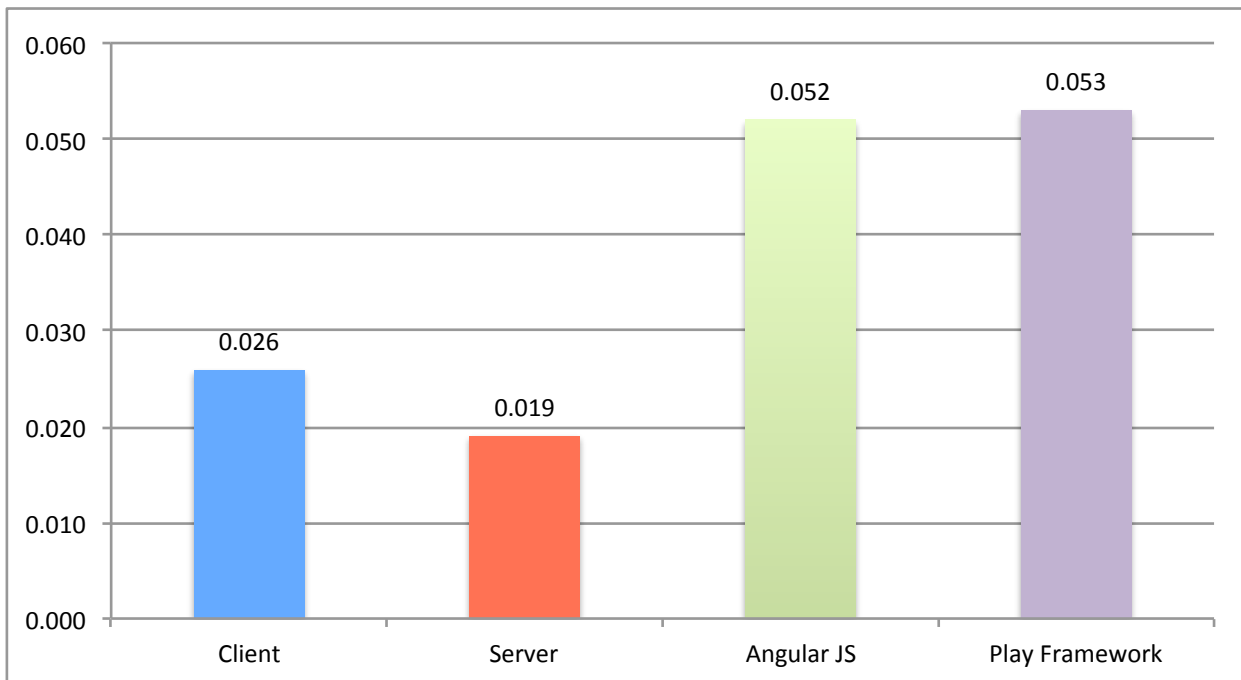


Abbildung B.19.: Halstead Bugs pro Methode

In Abbildung B.19 wird diese Komplexitätsgrösse mit zwei der durch EEPPI eingesetzten Frameworks verglichen. Sowohl der Client als auch der Server weisen einen deutlich besseren Wert aus als die beiden zum Vergleich aufgeführten Frameworks. Erstaunlich ist, dass die beiden Frameworks beinahe den gleichen Wert aufweisen.

Beim Anblick der Zahlen liegt die Vermutung nahe, dass es daran liegt, dass EEPPI einen geringeren Projektumfang hat als die beiden Frameworks. Es wird jedoch hier nicht die totale Grösse verglichen, sondern lediglich die Komplexität pro Methode. Diese Vermutung trifft deshalb nicht zu.

B.5.4.1. TypeScript

Die Ergebnisse werfen auch die Frage auf, was TypeScript bringt und ob es sich dessen Einsatz gelohnt hat.

Aus unserer Sicht hat sich der Einsatz von TypeScript sehr gelohnt. Viele Fehler haben wir dadurch bereits zur Kompilierzeit gefunden und nicht erst zur Laufzeit. Der Source Code ist strukturierter, modularisierter, gekapselter und trotzdem lesbarer als vergleichbarer JavaScript Code dies wäre. Selbst die Metriken, die anhand des erzeugten JavaScript Codes erstellt wurden, zeigen sehr positive Werte und bestätigten unseren subjektiven Eindruck.

C. Infrastruktur

C.1. Arbeitsplätze, Geräte und Server

Um entwickeln, builden und testen zu können, benötigt das Team die folgende Infrastruktur:

- Zugewiesene Arbeitsplätze im Zimmer 1.206
- Persönliche Entwicklungsgeräte für jedes Teammitglied, bevorzugt Laptop (eigene Geräte)
- Virtuelle Server für Projektmanagement und als Entwicklungsserver

C.2. Tools

C.2.1. Projektmanagement

Jira und Redmine bieten ähnliche Funktionalität und Teammitglieder haben mit Jira wie mit Redmine gute Erfahrungen gemacht. Redmine ist in der Basiskonfiguration eher auf RUP ausgerichtet, für Agile Entwicklung werden Plugins benötigt. Jira ist in der Basiskonfiguration auf Agile Entwicklung ausgerichtet. Die Benutzeroberfläche von Jira ist etwas moderner und benutzerfreundlicher gestaltet, ansonsten sind sich beide Oberflächen ähnlich.

Jira ist kostenpflichtig, kostet allerdings nur 10\$ [1] und wird für nicht kommerzielle Projekte sogar gratis angeboten. Vom Hersteller selbst angebotene Jira Plugins kosten häufig auch nur 10\$, sodass das Erweitern von Jira mit Plugins auch für kleine Projekte finanziell tragbar ist.

Redmine ist OpenSource und es existieren auch viele kostenlose Plugins.

Evaluierte Produkte Jira, Redmine

Ausgewähltes Produkt Jira

Begründung Jira bietet eine benutzerfreundliche Oberfläche, die erforderliche Funktionalität und ist grundsätzlich auf Agile Entwicklung ausgerichtet

C.2.2. Versionsverwaltung

C.2.2.1. Git

Ausgewähltes Produkt Git

Mögliche Alternativen SVN

Begründung Git ist ein weit verbreitetes Versionsverwaltungstool, das sich bei den Teammitgliedern bei privaten, schulischen und geschäftlichen Projekten bewährt hat. Git bietet den Vorteil von lokalen Repositories, ist wesentlich schneller als SVN, benötigt für vergleichbare Repositories spürbar weniger Platz und bringt eine gute Mergeautomatik mit.

C.2.2.2. GitHub

Evaluierte Produkte HSR Git, GitHub

Ausgewähltes Produkt GitHub

Mögliche Alternativen GitLab (Self hosted)

Begründung Mit GitHub besitzen die Studenten durch andere Projekte bereits Erfahrung. Als Studenten haben sie Zugriff auf kostenlose «Private-Repositories». Zudem bietet GitHub noch zusätzliche Funktionen wie Wiki, RST- und MD-Viewer sowie Repository-Zugriff und Dateibearbeitung über ein Webinterface.

C.2.2.3. Git Flow

Git Flow¹ automatisiert häufige genutzte Git Operationen für einen Entwicklungsworkflow [2] mit Master-, Develop-, und Featurebranches. Beispiel: Das Release einer neuen Version inklusive Branching, Merging und Tagging.

C.2.3. Literaturverwaltung

Zur Verwaltung von Literatur hat sich das Team für «Zotero»² entschieden. Zotero ist den Studenten seit der Einführung in Literaturrecherchen Anfang Studium bekannt und bietet viele praktische Funktionen, wie das Erzeugen von Literaturnachweisen aus aufgerufenen Webseiten. Es ist als Browser Plugin verfügbar, erlaubt das Teilen der Literaturliste unter den Teammitgliedern und bietet einen Export nach BibTex.

C.2.4. Dokumentation

C.2.4.1. Für grosse Dokumentationen und Abgabedokumente: \LaTeX

\LaTeX ist sehr gut geeignet für grosse, gemeinsam zu erarbeitende Dokumente, weil die Source-Dateien über Git versioniert und gemergt werden können und wenig Platz verbrauchen. Zudem besteht ein sehr kleines Risiko auf Dokumentenverlust bzw. Dokumentenfehler durch die Software, weil \LaTeX die Source-Dateien gar nicht verändert, im Unterschied zu einer Office-Applikation.

¹«Git Extensions zur Automatisierung von Repository Workflows»: <https://github.com/nvie/gitflow>

²Freie Quellenverwaltungssoftware vom «Roy Rosenzweig Center for History and New Media»: <https://www.zotero.org/>

C.2.4.2. Für Literatur und temporäre Dokumente: Dropbox

PDF-Dokumente sowie Office-Dokumente können nur schlecht versioniert werden mit Git, da sie für Git ein BLOB³ darstellen und das Repository mit jeder Version unnötig aufblähen. Daher werden solche Dokumente über einen Cloudshare ausgetauscht und erst eingecheckt, wenn sie keinen oder kaum mehr Änderungen unterliegen.

Evaluierte Produkte Owncloud HSR, Owncloud L.M., Dropbox

Ausgewähltes Produkt Dropbox

Begründung OwnCloud wäre die Präferenz des Betreuers sowie der Studenten. OwnCloud bietet den Vorteil, dass die Hoheit über die Daten bei den Studenten selbst liegt, bzw. beim Host (Die HSR im Falle der HSR Cloud).

Dropbox legt die Daten in der Amazon Cloud ab. Die Datenschutzbestimmungen erlauben den Betreibern nicht nur die Analyse und Verwendung der Daten, sondern auch die Weitergabe an Drittanbieter.

Der Owncloud Client bietet leider keine Möglichkeit, zwischen mehreren Clouds umzuschalten. Daher müsste für die Verwendung einer BA Cloud der persönliche CloudSync deaktiviert werden, was sehr unpraktisch ist. Aus diesem Grund hat sich das Team trotz der Bedenken seitens Datenschutzes für Dropbox anstelle von OwnCloud entschieden.

C.2.4.3. Für Notizen & Meetingprotokolle: Restructured Text (rst), Markdown (md)

Für Notizen und kleine Dokumente haben sich die Studenten für RST und MD entschieden, da deren Funktionalität für den Zweck vollständig ausreicht und im Zusammenhang mit den eingesetzten Tools einige Vorteile bieten. Sie sind schlank, bieten nur das notwendigste an Markup, können versioniert und gemergt werden, weil es nur Textfiles sind, und werden von «GitHub Document Preview» unterstützt.

C.2.4.4. Für Diagramme, Skizzen: OpenDocument

Für Anwendungsfälle, in denen die bereits genannten Dokumentformate nicht ausreichen, wird OpenDocument eingesetzt. Dabei wird berücksichtigt, dass es über Git nicht inkrementell versioniert und nicht gemergt werden kann.

C.2.5. Dokumentation des API

Das EEPPI API ist zentraler Teil der Bachelorarbeit und aus diesem Grund ist deren Dokumentation (siehe Abschnitt E.1) sehr wichtig. Insbesondere die Korrektheit und Aktualität der Schnittstellendokumentation sind entscheidend die Benutzung der Schnittstelle.

Die Dokumentation wird durch den Server generiert und in Form eines HTML Dokuments zur Verfügung gestellt. Detailliert beschrieben werden die Mechanismen hinter der API in Abschnitt 6.2 auf Seite 53.

³Binary Large Object: Binärdatenblock, der nicht aus Zeichen sondern binären Daten besteht

C.2.6. Code Dokumentation

C.2.6.1. Client

Für zentrale und komplexe Klassen wie zum Beispiel das Repository wird Typedoc⁴ zur Erzeugung einer Dokumentation aus dem TypeScript Code und dessen TSDoc Kommentaren verwendet.

C.2.7. Modelling

Ausgewähltes Produkt Astah

Mögliche Alternativen Umllet

Begründung Astah ist ein gutes, den Studenten bekanntes Tool. Es deckt den geforderten Funktionsumfang grosszügig ab und bietet Image- sowie PDF-Export.

C.2.8. UI Mockups

Für den Entwurf von UI Mockup verzichten wir auf ein Tool und benutzen Papier und Stifte.

C.2.9. Building & Testing

C.2.9.1. Buildserver

Ein Build-Server soll zum automatisierten Ausführen der Tests eingesetzt werden. Einige JS Testing Frameworks bieten eine Anbindungsmöglichkeit an Build Server.

Ausgewähltes Produkt Jenkins

Begründung Jenkins ist ein etabliertes und weit verbreitetes Produkt, das sich gut konfigurieren lässt und aus diesem Grund eingesetzt wird. Die Client-Tests können mit Hilfe von Selenium zusammen mit den Server-Tests durchgeführt werden.

C.2.9.2. Testing

Das Testkonzept und die verwendeten Frameworks und Tools werden im Abschnitt B.3 unter Projektmanagement beschrieben.

C.2.10. Individuelle Entwicklungsumgebungen

Jeder Entwickler verwendet seine eigene bevorzugte Entwicklungsumgebung. Um auch Interaktionen mit externen Systemen zu testen, haben wir sogenannte Vagrant⁵-Boxen

⁴<http://typedoc.io>

⁵Virtualisierungsautomatisierungslösung von HashiCorp für verschiedene Virtualisierungsumgebungen:
<http://www.vagrantup.com/>

erstellt. Dies sind Konfigurationen für virtuelle Maschinen, die wenige Bytes gross sind und mit einem einfachen Befehl komplette Systeme simulieren können. Für EEPPI haben wir je Vagrant-Umgebungen für folgende Systeme erstellt:

- CDAR
- ADRepo
- Projektplanungstools (Jenkins und Jira)
- komplettes EEPPI (inkl. Umsysteme)

C.3. Backup

C.3.1. Persönliche Entwicklungsgeräte

C.3.1.1. MacBook von Laurin Murer

Die Daten auf dem Rechner von Laurin Murer werden mehrstufig archiviert. Einerseits mit TimeMachine⁶ auf eine externe Festplatte und zusätzlich noch mit Wuala⁷ in die Cloud.

C.3.1.2. Linux Laptop von Tobias Blaser

Die Daten auf dem Rechner von Tobias Blaser werden mit BackInTime⁸ auf eine externe Festplatte archiviert.

C.3.2. Arbeitsplätze im Zimmer 1.206

Auf diesen Rechnern werden keine Daten abgelegt, dadurch müssen sie auch nicht in einem Backup eingeschlossen werden.

C.3.3. Virtual Server

Vom virtuellen Server werden lediglich einige Daten archiviert und zwar jene, welche von uns erstellt wurden. Diese Daten werden alle unter /root/to_backup referenziert. Dieser Ordner wird dann regelmässig auf das MacBook von Laurin Murer kopiert, wo es dann in seinem Backup eingeschlossen wird (siehe C.3.1.1).

C.3.4. Repository und Entwicklungsserver

Ein zusätzliches Backup des Projektrepositories ist nicht notwendig, da durch die Versionierung mit Git die komplette Versionshistorie bei jedem Teilnehmer vorhanden ist. Somit ist das gesamte Projekt dreifach abgelegt (bei den Entwicklern sowie bei GitHub) sowie auch in den entsprechenden Backups integriert.

⁶Backuplösung von Apple: <https://www.apple.com/chde/support/timemachine/>

⁷Schweizer Cloud Speicher Dienst: <https://www.wuala.com/>

⁸Rsync basierte Backuplösung für Linux: <http://backintime.le-web.org/>

EEPPI-Usermanual

Suggested Environment

- Ubuntu Server 14.04 32 Bit
- Latest Build of EEPPI

Installation

If you would like to install and run EEPPI using a script, have a look to the setup script of the vagrant box (project/vagrant/setup.sh).

To install EEPPI by hand, do the following steps. You can use the declared commands if you are using suggested environment:

1. Install unzip

```
apt-get install unzip
```

2. Install java 8

```
sudo add-apt-repository ppa:webupd8team/java  
sudo apt-get update  
sudo apt-get install oracle-java8-installer`
```

3. Install EEPPI

1. Unpack EEPPI-1.0.zip

```
sudo unzip EEPPI-1.0.zip -d /usr/local/bin/eeppi_zip/  
sudo mv /usr/local/bin/eeppi_zip/`ls -1 /usr/local/bin/eeppi_zip/  
| tail -n 1` /usr/local/bin/eeppi/  
sudo rmdir /usr/local/bin/eeppi_zip
```

2. Start EEPPI

```
sudo /usr/local/bin/eeppi/bin/eeppi -Dhttp.port=80 -  
DapplyDownEvolutions.documentation=true -  
DapplyEvolutions.documentation=true -  
DapplyDownEvolutions.default=false -DapplyEvolutions.default=true  
&
```

3. Create crontab to start EEPPI after reboot

```
( crontab -l 2>/dev/null | grep -Fv ntpdate ; printf -- "@reboot
sudo /usr/local/bin/eeppi/bin/eeppi -Dhttp.port=80 -
DapplyDownEvolutions.documentation=true -
DapplyEvolutions.documentation=true -
DapplyDownEvolutions.default=false -DapplyEvolutions.default=true
&\n" ) | crontab
```

4. EEPPI ist available at <http://localhost:80>

Configure Play

See [Play documentation: Productive configuration](#) for advanced configuration.

- Change Database to PostgreSQL database 'eeppi' running on localhost:
 1. create an `application.conf` in your preferred configuration directory containing the following lines:

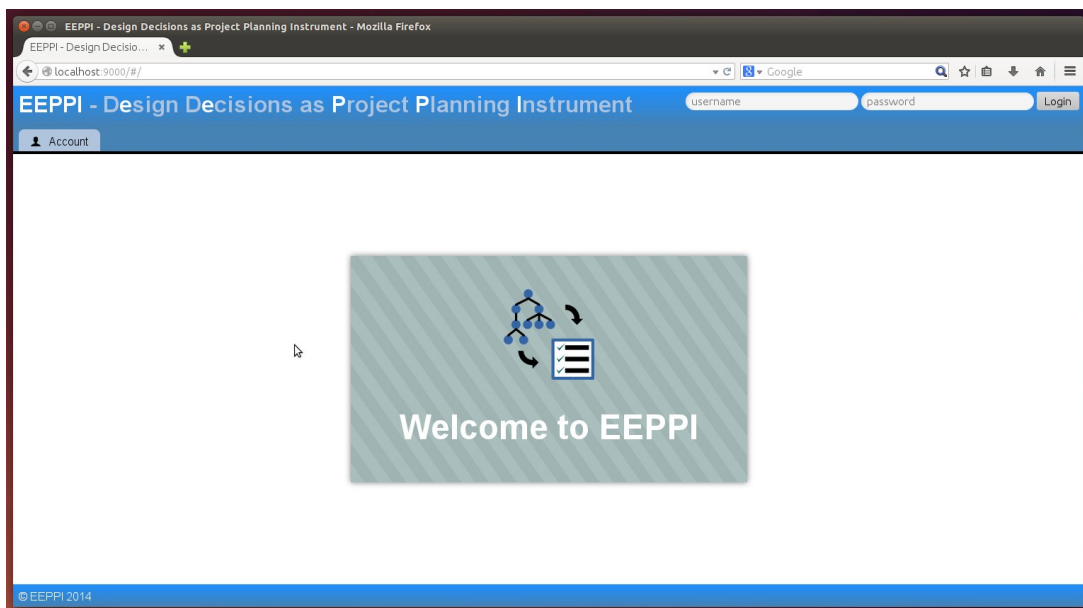
```
db.default.driver=org.postgresql.Driver
db.default.url="postgres://localhost/eeppi"
db.default.user="eeppiUser"
db.default.password="*****"
```

2. Run eeppi, specify alternative configuration file:

```
sudo /usr/local/bin/eeppi/bin/eeppi -Dhttp.port=80 -
DapplyDownEvolutions.documentation=true -
DapplyEvolutions.documentation=true -
DapplyDownEvolutions.default=false -DapplyEvolutions.default=true
-Dconfig.resource=/path/to/alternative/application.conf &
```

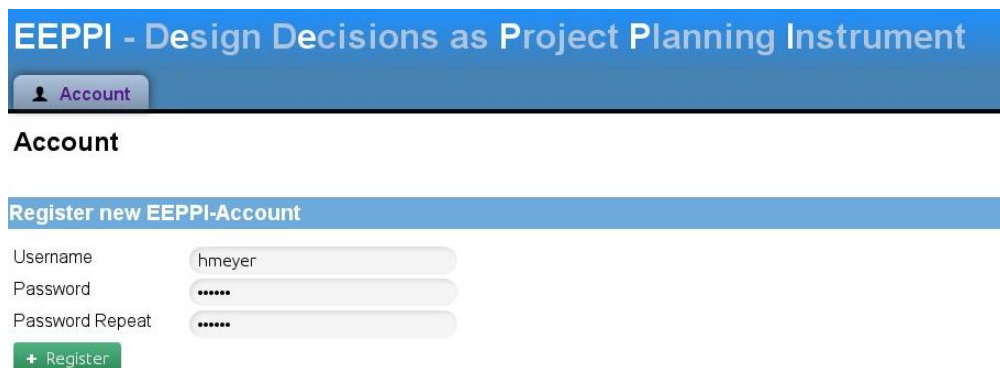
Configure EEPPI

1. Start EEPPI at <http://localhost:80>

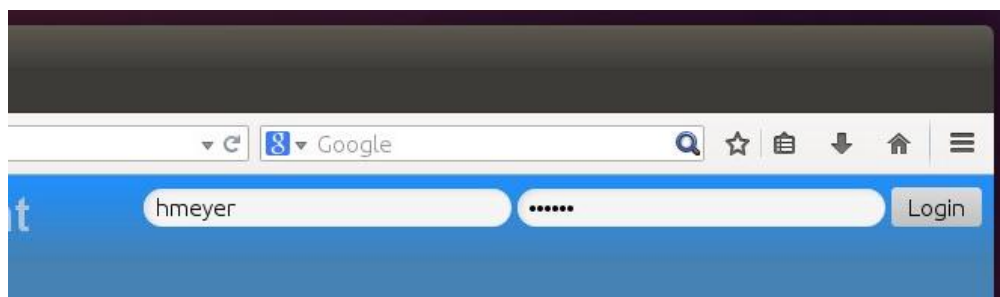


2. Create a new user account

1. Navigate to "Account"
2. Enter username & password and "Register" the new user.



3. Login with the just created user account



3. Define your Decision Knowledge System (e.g. your ADRepo)

1. Navigate to "Administration" > "Decision Knowledge Systems"
2. Enter the address of your DKS, changes will be applied on clicking outside field



Administration

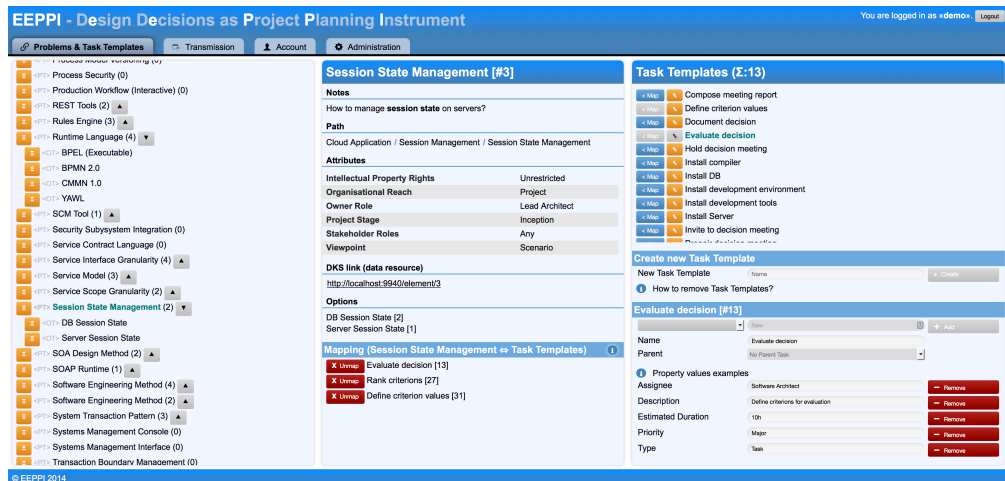
Manage properties, systems and requests.

Manage Decision Knowledge Systems (DKS)

A DKS is the source of all your Problems and Options. It is only read, nothing is written to it. In this version of EEPPi only one DKS is supported, however you can change it's URL here.

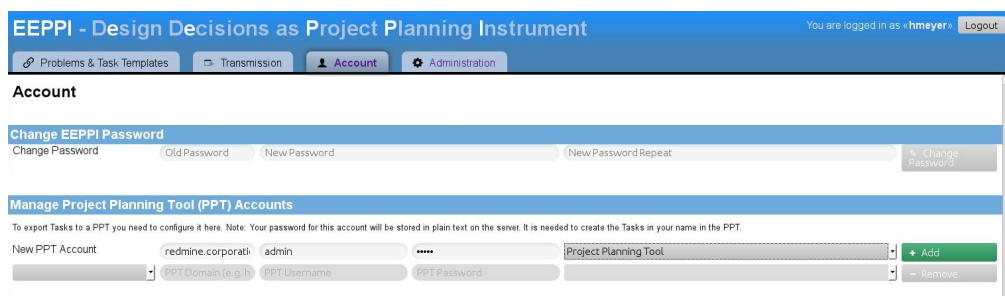
CorporationDKS | CorporationDKS | eeppi.corporation.com

- Navigate to "Problems & Task Templates" and check, if the items from your DKS will load to verify your configuration.



- Create an account for your Project Planning tool

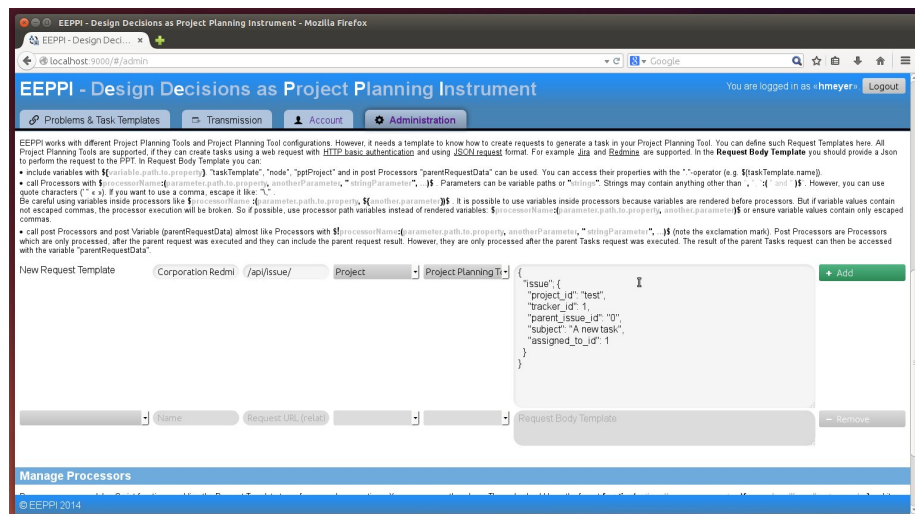
- Navigate to "Account" > "Project Planning Tool Accounts"
- Create a new account entering the url, username, password and project planning tool



- Create a template to transmit tasks to your project planning tool

- Navigate to "Administration" > "Request Templates"
- Create a new template entering api path and request body
 - Take a look into the documentation of the api of your project planning tool. E.g. [Redmine API: Creating an issue.](#)
 - Create a request to create an issue in your project planning tool. E.g. for Redmine:

```
{
  "issue": {
    "project_id": "test",
    "tracker_id": 1,
    "parent_issue_id": "0",
    "subject": "A new task",
    "assigned_to_id": 1
  }
}
```



3. Replace values by variables:

```
{
  "issue": {
    "project_id": "${pptProject}",
    "tracker_id": ${taskTemplate.type},
    "parent_issue_id": "${!parentRequestData.issue.id}",
    "subject": "${taskTemplate.name}",
    "assigned_to_id":
    ${taskTemplate.attributes.Assignee}
  }
}
```

4. Write the processors you need. Processors are JavaScript functions transforming values. E.g. tracker-transformation-processor:

1. Navigate to "Administration" > "Processors"
2. create the new Processor 'trackerTransformation':

```
function(typeValue) {
  if(typeValue == 'Feature') {
    return 1;
  } else if(typeValue == 'Bug') {
    return 2;
  } else {
    return 0;
  }
}
```

5. Use the processor transforming types to tracker ids:

```
"tracker_id": $trackerTransformation:(taskTemplate.type)$,
```

- Path values like 'taskTemplate.type' will be injected with object values:
 - 'taskTemplate': current task template object
 - 'node': current node from DKS system
 - 'pptProject': project identifier from transformation wizard
 - 'parentRequestData': only set if there was a parent request. Contains data returned from your project planning tool.
- Text values like "Text value" will be handled as text.
- You can use variables to create path or text values, e.g.

```
$someProcessor:
(${taskTemplate.attributes.nodeSpecialValuePath},
"This will be assigned to
${taskTemplate.attributes.Assignee}")$
```

- Escape commas inside processor arguments. Otherwise they will be interpreted as argument divider:

```
$otherProcessor:(("This will be assigned to
${taskTemplate.attributes.Assignee}\,
${taskTemplate.attributes.Stakeholder}")$
```

- There are two types of processors and variables:
 - '\$[..] variables and \$!:(..) processors': Will be executed before transmitting the request, you can not use 'parentRequestData' because it's not yet set
 - '\$![..] variables and \$!!(..) processors': Will be executed on transmitting the request, you can only use

'parentRequestData' to access the return values of the last parent request.

6. Use processors to generate JSON code:

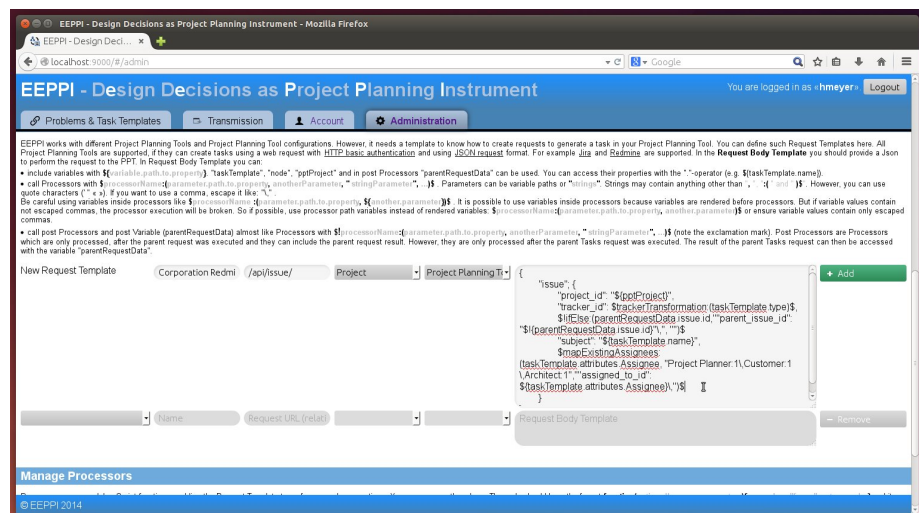
```
$!ifElse:(parentRequestData.issue.id,""parent_issue_id": "$!
{parentRequestData.issue.id}"\",", "")$
```

7. Use predefined processors like 'mapExistingAssignees':

```
$mapExistingAssignees:(taskTemplate.attributes.Assignee,
"Project
Planner:1\,Customer:1\,Architect:1",""assigned_to_id":
${taskTemplate.attributes.Assignee}\,")$
$replaceTaskTemplateValueByPPTValue:
(taskTemplate.attributes.Type,
"Bug:1\,Feature:2\,Support:3",""tracker_id":
${taskTemplate.attributes.Type}\,")$
```

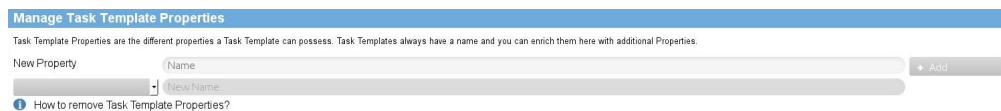
8. Complete your request template transforming all needed values. E.g.

```
{
  "issue": {
    "project_id": "${pptProject}",
    "tracker_id": $trackerTransformation:
(taskTemplate.type)$,
    $!ifElse:
(parentRequestData.issue.id,""parent_issue_id": "$!
{parentRequestData.issue.id}"\",", "")$
    "subject": "${taskTemplate.name}",
    $mapExistingAssignees:
(taskTemplate.attributes.Assignee, "Project
Planner:1\,Customer:1\,Architect:1",""assigned_to_id":
${taskTemplate.attributes.Assignee}\,")$
  }
}
```

6. Create task template properties

1. Navigate to "Administration" > "Manage Task Template Properties"



2. Add the new properties you need, e.g. "Stakeholder"

3. Modify existing properties. You can't remove properties but you can rename them.

7. Create task templates

1. Navigate to "Problems & Task Templates" > "Task Templates"

Task Templates (Σ:13)

- < Map Compose meeting report
- < Map Define criterion values
- < Map Document decision
- < Map **Evaluate decision**
- < Map Hold decision meeting
- < Map Install compiler
- < Map Install DB
- < Map Install development environment
- < Map Install development tools
- < Map Install Server
- < Map Invite to decision meeting
- < Map Review decision meeting

Create new Task Template

New Task Template + Create

i How to remove Task Templates?

Evaluate decision [#13]

+ Add

Name

Parent

i Property values examples

Assignee - Remove

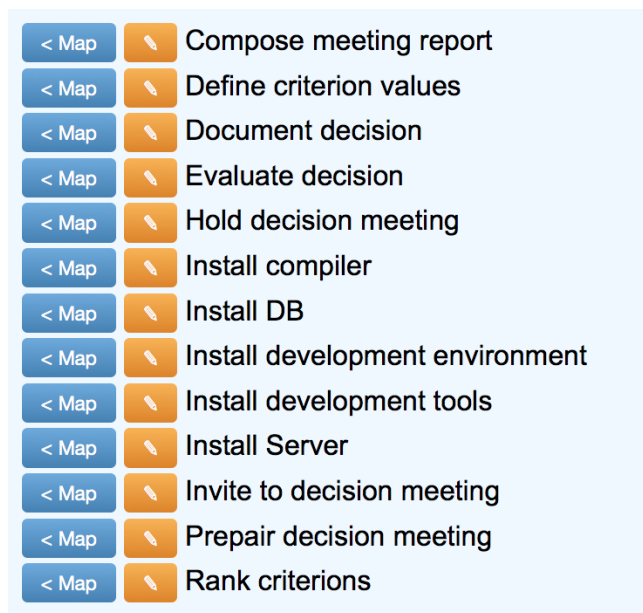
Description - Remove

Estimated Duration - Remove

Priority - Remove

Type - Remove

1. Add new Task Templates
 1. Navigate to "Create new Task Template"
 2. Enter the name of the new template, e.g. "Install debug tools", save it.
 3. Change the properties and property values below
2. Map task templates with problems
 1. Select the problem or alternative on the left
 2. "Map" the task template on the right



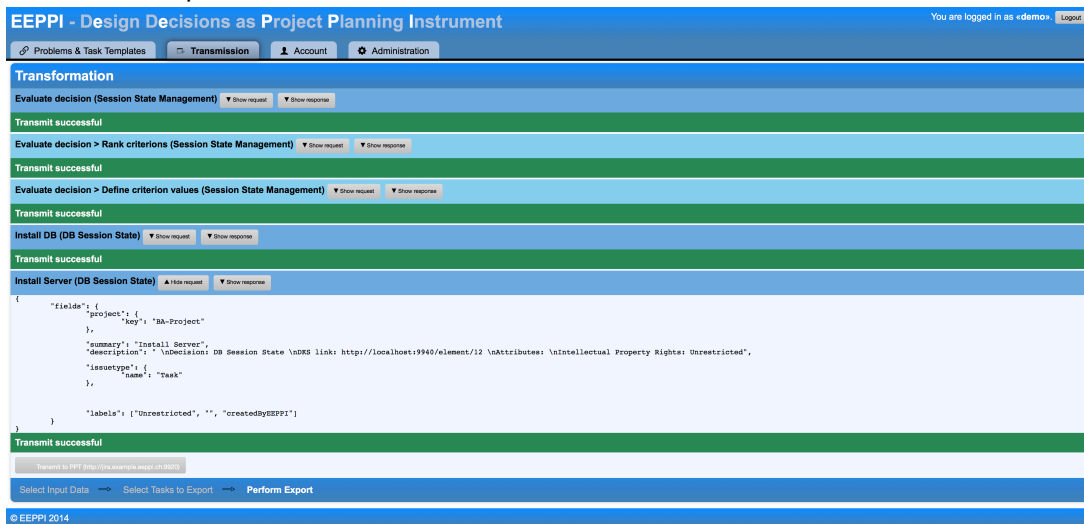
8. Transfer tasks to a project planning tool

1. Navigate to "Transmission"
2. Choose the target system (your Redmine or Jira the tasks should be exported to)

1. Choose the request template to transmit the tasks. Please ensure you choose the correct request template matching your target system. If your target system is Redmine, be sure you are using a Redmine request template.
2. Fill in the identifier of the project in your project planning tool. This depends how you mapped this value in the request template.
3. Choose the tasks you would like to export.

Problem	Decision	Alternative	Export?	Parent?	Task
DB Model [#6]	DB Model [#16]				Hold decision meeting [#36]
Session State Management [#3]	Session State Management [#14]		<input checked="" type="checkbox"/>		Evaluate decision [#13]
Session State Management [#3]	Session State Management [#14]		<input checked="" type="checkbox"/>	Evaluate decision	Rank criterions [#27]
Session State Management [#3]	Session State Management [#14]		<input checked="" type="checkbox"/>	Evaluate decision	Define criterion values [#31]
Session State Management [#3]	Session State Management [#14]	DB Session State [#12]	<input checked="" type="checkbox"/>	Not as a Subtask	Install DB [#21]
Session State Management [#3]	Session State Management [#14]	DB Session State [#12]	<input checked="" type="checkbox"/>	Not as a Subtask	Install Server [#24]

1. Set the parent of subtasks or tasks of an option of a decisions. You can't create hierarchies with more than one sublayer, because some project planning system do not support this.
2. Transform the tasks and check the generated syntax of the request at minimum before the first export. The syntax should be a correct json containing all your values and there may be some secondary variables and processors (starting with '\$!') if you used some. This placeholders will be replaced on transmitting the template.



1. If the syntax is correct and your project planning tool running, start the transmission. After the transmit, you can see what data your project planning tool returned and use this data for more advanced secondary processors in your request template, e.g. to map tasks with parent tasks. If the transmit failed, the response can help you to determine the problem. Most problems sources are:
 - JSON syntax incorrect
 - JSON is not matching the API of your project planning tool correctly (maybe incorrect field names or values)
 - Project planning tool is not running
9. Login to your project planning tool and check the created tasks.

D.1. Beispieldaten für Request-Templates

Wie im Usermanual beschrieben, braucht es ein Request-Template, um Tasks an ein Projektplanungstool zu übertragen. In Abbildung D.1 ist ein Beispiel für ein Request-Template abgebildet, das gleiche ist auch im Usermanual zu finden.

```
1 {
2   "issue": {
3     "project_id": "${pptProject}",
4     "tracker_id": $getTrackerByType:(taskTemplate.type)$,
5     "subject": "${taskTemplate.name}",
6     "assigned_to_id": $getAssigneeIdByName:(taskTemplate.attributes.
       Assignee)$
7   }
8 }
```

Abbildung D.1.: Beispiel eines Request-Tempaltes

In diesem Beispiel ist zu sehen, wie verschiedene Variablen verwendet werden. Es sind die folgenden Variablen verfügbar, sie sind auch im Usermanual beschrieben:

- taskTemplate
- node
- pptProject
- parentRequestData

Diese Variablen repräsentieren Objekte, beziehungsweise ganze Objektgraphen. Mittels dem Punkt-Operator kann durch diesen Graph navigiert werden.

Nachfolgend wird ein Beispiel eines Objektgraphen gezeigt, um eine Vorstellung zu vermitteln, welche Daten verfügbar sind.

```
1 /* "taskTemplate" and "node" example data for usage in variables and processors */
2
3 {
4     "taskTemplate": {
5         "id": 158,
6         "name": "Define criteria",
7         "properties": [
8             {
9                 "id": 159,
10                "property": {
11                    "id": 152,
12                    "name": "Assignee"
13                },
14                "value": "Project Planner"
15            },
16            {
17                "id": 160,
18                "property": {
19                    "id": 153,
20                    "name": "Type"
21                },
22                "value": "Task"
23            },
24            {
25                "id": 161,
26                "property": {
27                    "id": 154,
28                    "name": "Description"
29                },
30                "value": "Define criteria for evaluation"
31            },
32            {
33                "id": 162,
34                "property": {
35                    "id": 155,
36                    "name": "Priority"
37                },
38                "value": "Major"
39            },
40            {
41                "id": 163,
42                "property": {
43                    "id": 156,
44                    "name": "Due Date"
45                },
46                "value": "2015-01-14"
47            },
48            {
49                "id": 164,
50                "property": {
51                    "id": 157,
52                    "name": "Estimated Duration"
53                },
54                "value": "10h"
55            }
56        ],
57        "parent": null,
58        "attributes": {
59            "Assignee": "Project Planner",
60            "Type": "Task",
61            "Description": "Define criteria for evaluation",
62            "Priority": "Major",
63            "Due Date": "2015-01-14",
64            "Estimated Duration": "10h"
65        }
66    },
67    "node": {
```

```
68     "id": 19,
69     "name": "Service Model",
70     "path": ["AVT", "Cloud Providers", "Service Model"],
71     "attributes": {
72         "Revision Date": "2016-11-11",
73         "Viewpoint": "Scenario",
74         "Intellectual Property Rights": "Unrestricted",
75         "Due Date": "2014-12-24",
76         "Project Stage": "Inception",
77         "Organisational Reach": "Project",
78         "Stakeholder Roles": "Any",
79         "Owner Role": "Lead Architect"
80     },
81     "notes": "Which xaaS Service Model ... \r\n\r\nSee CCP website for ...",
82     "self": "http://localhost:9940/element/19",
83     "alternatives": [
84         {
85             "id": 17,
86             "name": "IaaS",
87             "path": ["AVT", "Cloud Providers", "IaaS"],
88             "attributes": {
89                 "Intellectual Property Rights": "Unrestricted"
90             },
91             "notes": "http://www.cloudcomputingpatterns.org/...",
92             "self": "http://localhost:9940/element/17",
93             "template": {
94                 "id": 7,
95                 "name": "IaaS",
96                 "path": ["Cloud Application", "Cloud Providers", "IaaS"],
97                 "attributes": null,
98                 "notes": null,
99                 "self": "http://localhost:9940/element/7"
100            },
101            "state": "Eligible"
102        },
103        {
104            "id": 18,
105            "name": "PaaS",
106            "path": ["AVT", "Cloud Providers", "PaaS"],
107            "attributes": {
108                "Intellectual Property Rights": "Unrestricted"
109            },
110            "notes": "http://www.cloudcomputingpatterns.org/PaaS\r\n\r\n...",
111            "self": "http://localhost:9940/element/18",
112            "template": {
113                "id": 8,
114                "name": "PaaS",
115                "path": ["Cloud Application", "Cloud Providers", "PaaS"],
116                "attributes": null,
117                "notes": null,
118                "self": "http://localhost:9940/element/8"
119            },
120            "state": "Neglected"
121        }
122    ],
123     "template": {
124         "id": 10,
125         "name": "Service Model",
126         "path": ["Cloud Application", "Cloud Providers", "Service Model"],
127         "attributes": null,
128         "notes": null,
129         "self": "http://localhost:9940/element/10"
130     },
131     "state": "PartiallySolved"
132 },
133 "pptProject": "test"
134 }
```

E. Code-Documentation

E.1. API

API Documentation for EEPPI

1 Authentication

2 Parameters

3 Resources

3.1 Decision Knowledge System

- **POST /rest/api/1/dks** (Stores a new DKS (Decision Knowledge System).)
- **GET /rest/api/1/dks** (Returns all DKSs (Decision Knowledge Systems).)
- **POST /rest/api/1/dks/<id>** (Updates a DKS (Decision Knowledge System).)
- **GET /rest/api/1/dks/<id>** (Returns one DKS (Decision Knowledge System).)
- **DELETE /rest/api/1/dks/<id>** (Deletes a DKS (Decision Knowledge System).)
- **GET /rest/api/1/dks/getFromDKS?url=<url>** (Redirects a request to a remote server using GET to avoid restrictions with Cross Origin Requests.)

3.2 Decision Knowledge System Mapping

- **POST /rest/api/1/dksMapping** (Creates a new Mapping for a DKS Node and a Task Template.)
- **GET /rest/api/1/dksMapping** (Reads all Mappings for DKS Nodes and Task Templates.)
- **POST /rest/api/1/dksMapping/<id>** (Updates an existing Mapping for a DKS Node and a Task Template.)
- **GET /rest/api/1/dksMapping/<id>** (Reads a Mapping for a DKS Node and a Task Template.)
- **DELETE /rest/api/1/dksMapping/<id>** (Deletes a Mapping for a DKS Node and a Task Template.)
- **GET /rest/api/1/dksMapping/byDKSNode/<dksNode>** (Reads all Mappings for a given DKS Node.)

3.3 Processor

- **POST /rest/api/1/processor** (Persists the given processor)
- **GET /rest/api/1/processor** (Returns all processors.)
- **POST /rest/api/1/processor/<id>** (Updates a processor.)
- **GET /rest/api/1/processor/<id>** (Returns one processor.)
- **DELETE /rest/api/1/processor/<id>** (Deletes a processor.)

3.4 Project Planning Tool

- **GET /rest/api/1/ppt** (Returns all Project Planning Tools.)
- **GET /rest/api/1/ppt/<id>** (Returns one Project Planning Tool.)
- **POST /rest/api/1/ppt/createPPTTask** (Creates a Task on a remote Project Planning Tool Server and stores the creation on the server.)

3.5 Request Template

- **POST /rest/api/1/requestTemplate** (Stores a new Request Template for sending a Task to a Project Planning Tool.)
- **GET /rest/api/1/requestTemplate** (Returns all Request Templates for sending a Task to a Project Planning Tool.)
- **POST /rest/api/1/requestTemplate/<id>** (Updates a Request Template for sending a Task to a Project Planning Tool.)
- **GET /rest/api/1/requestTemplate/<id>** (Returns one Request Template for sending a Task to a Project Planning Tool.)
- **DELETE /rest/api/1/requestTemplate/<id>** (Deletes a Request Template for sending a Task to a Project Planning Tool.)

3.6 Task Property

- **POST** `/rest/api/1/taskProperty` (Creates a new Task Property.)
- **GET** `/rest/api/1/taskProperty` (Reads all Task Properties.)
- **POST** `/rest/api/1/taskProperty/<id>` (Updates an existing Task Property with new data.)
- **GET** `/rest/api/1/taskProperty/<id>` (Reads a Task Property.)
- **DELETE** `/rest/api/1/taskProperty/<id>` (Deletes a Task Property.)

3.7 Task Template

- **POST** `/rest/api/1/taskTemplate` (Creates a new Task Template of which (concrete) Tasks then can be generated.)
- **GET** `/rest/api/1/taskTemplate` (Reads all Task Templates.)
- **POST** `/rest/api/1/taskTemplate/<id>` (Updates an existing Task Template with new data.)
- **GET** `/rest/api/1/taskTemplate/<id>` (Reads a Task Template.)
- **DELETE** `/rest/api/1/taskTemplate/<id>` (Deletes a Task Template.)
- **POST** `/rest/api/1/taskTemplate/<id>/addProperty` (Adds a new property to an existing Task Template.)
- **POST** `/rest/api/1/taskTemplate/<id>/properties/<taskTemplate>` (Updates a task property value.)
- **DELETE** `/rest/api/1/taskTemplate/<id>/properties/<taskTemplate>` (Deletes a task property value.)

3.8 PPTAccount

- **POST** `/rest/api/1/user/pptAccount` (Stores a new login information for a Project Planning Tool.)
- **GET** `/rest/api/1/user/pptAccount` (Returns all login information (but the password) for the currently logged in user for Project Planning Tools.)
- **POST** `/rest/api/1/user/pptAccount/<id>` (Updates login information for a Project Planning Tool on the server.)
- **GET** `/rest/api/1/user/pptAccount/<id>` (Returns one login information (but the password) for the currently logged in user for Project Planning Tools.)
- **DELETE** `/rest/api/1/user/pptAccount/<id>` (Deletes login information for a Project Planning Tool on the server.)

3.9 Project

- **GET** `/rest/api/1/project` (Returns all Projects.)
- **GET** `/rest/api/1/project/<id>` (Returns one Project.)

3.10 User

- **POST** `/rest/api/1/user/changePassword` (This changes the password of an EEPPi-user.)
- **POST** `/rest/api/1/user/login` (Checks the login information for the user and if the login is successful a cookie is set.)
- **GET** `/rest/api/1/user/loginStatus` (Returns the login status for the currently logged in user and a Json representation of it.)
- **POST** `/rest/api/1/user/logout` (Does log out the currently logged in user by removing the cookie.)
- **POST** `/rest/api/1/user/register` (This creates a new EEPPi-user.)

1 Authentication

Some methods need authentication. The authentication can be made by providing a cookie generated by [/rest/api/1/user/login](#) or by providing HTTP Basic Authentication. The HTTP Basic Authentication could be used with something like <http://username:password@localhost:9000/rest/api/1/user/pptAccount?basicAuth=true>. The GET parameter "basicAuth=true" lets the server enable HTTP Basic Authentication. A response with status code 401 (Unauthorized) is returned, when authentication information are required but are not provided.

2 Parameters

All method parameters can be passed in two ways:

- Normal POST-data (this is also used in this documentation):

```
curl --request POST --data "theKey=theValue&theKey2=theValue2"
http://localhost:9000/...
```

- As a JSON-object as binary data:

```
curl -H 'Content-Type: application/json;charset=UTF-8' --data-binary
'{"theKey":"theValue","theKey2":"theValue2"}' http://localhost:9000/...
```

If it's passed as a JSON-object and some parameter reference another object, the reference can be passed in two ways:

- As a numeric ID of the referenced object:

```
curl -H 'Content-Type: application/json;charset=UTF-8' --data-binary
'{"id":61,"name":"Example processor","project":63,"code":"function(num) {
return num*num; }"}' http://localhost:9000/rest/api/1/processor
```

- As a full object containing at least an ID-property:

```
curl -H 'Content-Type: application/json;charset=UTF-8' --data-binary
'{"id":61,"name":"Example processor","project":{"id":63,"name":"The Example
Project"},"code":"function(num) { return num*num; }"}' http://localhost:9000
/rest/api/1/processor
```

If it's passed as a full object and the method returns the referenced object again, it's returned as it got passed. However, it's not updated on the server.

3 Resources

3.1 Decision Knowledge System

POST /rest/api/1/dks

Stores a new DKS (Decision Knowledge System).

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
name	String	the new name of the new DKS
url	String	the URL where the DKS is

Responses

Status	Description
400	If the DKS could not be stored.
200	If the DKS was stored. It is also returned in Json form.

Example 1

```
curl --request POST --data "name=The DKS&url=" http://localhost:9000/rest/api/1/dks
```

The previous command **did** return with status code **500 (Internal Server Error)** and yielded the following:

```
This is not implemented yet, it is a known limitation of this version.
```

Example 2

```
curl --request POST --data "name=The DKS&url=http://the-dks.ch" http://localhost:9000/rest/api/1/dks
```

The previous command **did** return with status code **500 (Internal Server Error)** and yielded the following:

```
This is not implemented yet, it is a known limitation of this version.
```

GET /rest/api/1/dks

Returns all DKSs (Decision Knowledge Systems).

[Authentication](#) is required to use this method.

Responses

Status	Description
200	It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/dks
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{ "items": [ { "id": 1000000000000000071, "name": "Example DKS", "url": "http://an-example-dks.com" }, { "id": 1000000000000000073, "name": "Example DKS", "url": "http://an-example-dks.com" }, { "id": 1000000000000000074, "name": "Example DKS", "url": "http://an-example-dks.com" }, { "id": 1000000000000000075, "name": "Example DKS", "url": "http://an-example-dks.com" } ] }
```

POST /rest/api/1/dks/<id>

Updates a DKS (Decision Knowledge System).

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the mapping to update
name	String	the new name of the new DKS to update
url	String	the URL where the DKS is

Responses

Status	Description
404	If no entity with the given ID exists.
400	If the request parameter contain errors.
200	The new created entity is returned

Example 1

```
curl --request POST --data "name=Example DKS&url=http://a-dks.com"
http://localhost:9000/rest/api/1/dks/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find DecisionKnowledgeSystem with id 9999."
```

Example 2

```
curl --request POST --data "name=Example DKS&url=http://an-example-dks.com"
http://localhost:9000/rest/api/1/dks/10000000000000000073
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":10000000000000000073,"name":"Example DKS","url":"http://an-example-
dks.com"}
```

Example 3

```
curl --request POST --data "name=Example DKS&url=" http://localhost:9000
/rest/api/1/dks/10000000000000000074
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"url":{"This field is required"}}
```

GET /rest/api/1/dks/<id>

Returns one DKS (Decision Knowledge System).

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to get

Responses

Status	Description
404	If no entity with the given ID exists.
200	If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/dks/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find DecisionKnowledgeSystem with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/dks/10000000000000000071
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":10000000000000000071,"name":"Example DKS","url":"http://an-example-
dks.com"}
```

DELETE /rest/api/1/dks/<id>

Deletes a DKS (Decision Knowledge System).

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to delete

Responses

Status	Description
404	If the DKS to delete could not be found.
204	If the DKS was deleted.

Example 1

```
curl --request DELETE http://localhost:9000/rest/api/1/dks/9999
```

The previous command **did** return with status code **500 (Internal Server Error)** and yielded the following:

```
This is not implemented yet, it is a known limitation of this version.
```

Example 2

```
curl --request DELETE http://localhost:9000/rest/api/1/dks/1000000000000000075
```

The previous command **did** return with status code **500 (Internal Server Error)** and yielded the following:

```
This is not implemented yet, it is a known limitation of this version.
```

GET /rest/api/1/dks/getFromDKS?url=<url>

Redirects a request to a remote server using GET to avoid restrictions with Cross Origin Requests.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
url	String	The full URL of the remote server to GET from.

Responses

Status	Description
400	If there is an error during preparation of the request for the remote server.
-	The return value from the remote server is returned.

Example 1

```
curl http://localhost:9000/rest/api/1/dks/getFromDKS?url=http://headers.jsontest.com/
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"Host": "headers.jsontest.com", "User-Agent": "NING/1.0", "Accept": "*/*"}
```

Example 2

```
curl http://localhost:9000/rest/api/1/dks/getFromDKS?url=hatetepe?__no-valid-url
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
Could not load hatetepe?__no-valid-url
```

3.2 Decision Knowledge System Mapping**POST /rest/api/1/dksMapping**

Creates a new Mapping for a DKS Node and a Task Template.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
taskTemplate	String	The Task Template to map to a DKS Node
dksNode	String	The DKS Node to map to a Task Template

Responses

Status	Description
400	If the request parameter contain errors.
200	The new created entity is returned.

Example 1

```
curl --request POST --data "taskTemplate=10000000000000000033&dksNode=80" http://localhost:9000/rest/api/1/dksMapping
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":3353,"taskTemplate":{"id":10000000000000000033,"properties": [{"parent":null,"name":"My example Task Template"}],"dksNode":"80"}
```

Example 2

```
curl --request POST --data "taskTemplate=9999&dksNode=87" http://localhost:9000/rest/api/1/dksMapping
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"taskTemplate":["Invalid value"]}
```

GET /rest/api/1/dksMapping

Reads all Mappings for DKS Nodes and Task Templates.

[Authentication](#) is required to use this method.

Responses

Status	Description
200	It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/dksMapping
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{
  "items": [
    {
      "id": "3308",
      "taskTemplate": {
        "id": "3307",
        "properties": [
          {
            "parent": null,
            "name": "My example Task Template 7"
          }
        ],
        "dksNode": "1000000000000000064"
      },
      "id": "3353",
      "taskTemplate": {
        "id": "1000000000000000033",
        "properties": [
          {
            "parent": null,
            "name": "My example Task Template"
          }
        ],
        "dksNode": "80"
      },
      "id": "1000000000000000035",
      "taskTemplate": {
        "id": "3303",
        "properties": [
          {
            "parent": null,
            "name": "My example Task Template 2"
          }
        ],
        "dksNode": "87"
      },
      "id": "1000000000000000037",
      "taskTemplate": {
        "id": "3301",
        "properties": [
          {
            "parent": null,
            "name": "My example Task Template 2"
          }
        ],
        "dksNode": "87"
      },
      "id": "1000000000000000039",
      "taskTemplate": {
        "id": "3305",
        "properties": [
          {
            "parent": null,
            "name": "My example Task Template 2"
          }
        ],
        "dksNode": "87"
      }
    }
  ]
}
```

POST /rest/api/1/dksMapping/<id>

Updates an existing Mapping for a DKS Node and a Task Template.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the Mapping to update
taskTemplate	String	The Task Template to map to a DKS Node
dksNode	String	The DKS Node to map to a Task Template

Responses

Status	Description
404	If no entity with the given ID exists.
400	If the request parameter contain errors.

200 The new created entity is returned

Example 1

```
curl --request POST --data "taskTemplate=1000000000000000033&dksNode=87" http://localhost:9000/rest/api/1/dksMapping/1000000000000000037
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{
  "id": "1000000000000000037",
  "taskTemplate": {
    "id": "1000000000000000033",
    "properties": [
      {
        "parent": null,
        "name": "My example Task Template"
      }
    ],
    "dksNode": "87"
  }
}
```

Example 2

```
curl --request POST --data "taskTemplate=9999&dksNode=87" http://localhost:9000/rest/api/1/dksMapping/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Decision Knowledge System Mapping with id 9999."
```

GET /rest/api/1/dksMapping/<id>

Reads a Mapping for a DKS Node and a Task Template.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to get

Responses

Status	Description
404	If no entity with the given ID exists.
200	If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/dksMapping/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the

following:

```
"Could not find Decision Knowledge System Mapping with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/dksMapping/1000000000000000035
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":1000000000000000035,"taskTemplate":{"id":3303,"properties":
[],"parent":null,"name":"My example Task Template 2"},"dksNode":"87"}
```

DELETE /rest/api/1/dksMapping/<id>

Deletes a Mapping for a DKS Node and a Task Template.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to delete

Responses

Status	Description
404	If no entity with the given ID exists.
409	If the entity could not be deleted.
204	If the entity is successfully deleted.

Example 1

```
curl --request DELETE http://localhost:9000/rest/api/1/dksMapping/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Decision Knowledge System Mapping with id 9999."
```

Example 2

```
curl --request DELETE http://localhost:9000/rest/api/1/dksMapping
/1000000000000000039
```

The previous command **did** return with status code **204 (No Content)** and yielded the following:

GET /rest/api/1/dksMapping/byDKSNode/<dksNode>

Reads all Mappings for a given DKS Node.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
dksNode	String	The id of the DKS Node to get the mappings for

Responses

Status	Description
200	A list of all Mappings is returned, if no mapping could be found, an empty list is returned.

Example 1

```
curl http://localhost:9000/rest/api/1/dksMapping/byDKSNode/9999
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"items":[]}
```

Example 2

```
curl http://localhost:9000/rest/api/1/dksMapping/byDKSNode/1000000000000000064
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"items":[{"id":3308,"taskTemplate":{"id":3307,"properties":
[],"parent":null,"name":"My example Task Template
7"},"dksNode":"1000000000000000064"}]}
```

3.3 Processor

POST /rest/api/1/processor

Persists the given processor

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
name	String	a speaking name to identify the processor
project	String	a reference (ID) to the Project

code	String	JavaScript code of the processor
------	--------	----------------------------------

Responses

Status	Description
400	If the processor could not be stored.
200	If the processor was stored successfully. The processor is also returned as JSON object.

Example 1

```
curl --request POST --data "name=Example processor&project=9999&
code=function(num) { return num*num; }" http://localhost:9000/rest/api
/1/processor
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"project":{"Invalid value"}}
```

Example 2

```
curl --request POST --data "name=Example processor&project=1000000000000000042&
code=function(num) { return num*num; }" http://localhost:9000/rest/api
/1/processor
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":3354,"name":"Example processor","project":
{"id":1000000000000000042,"name":"The Example Project"},"code":"function(num) {
return num*num; }"}
```

GET /rest/api/1/processor

Returns all processors.

[Authentication](#) is required to use this method.

Responses

Status	Description
200	It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/processor
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{
  "items": [
    {
      "id": 3354,
      "name": "Example processor",
      "project": "The Example Project",
      "code": "function(num) { return num*num; }"
    },
    {
      "id": 10000000000000000042,
      "name": "The Example Project",
      "code": "function(num) { return num*num; }"
    },
    {
      "id": 10000000000000000054,
      "name": "Example Processor",
      "project": "Example project",
      "code": "function(a) { return a+'.'+a; }"
    },
    {
      "id": 10000000000000000055,
      "name": "Example Processor",
      "project": "Example project",
      "code": "function(a) { return a+'.'+a; }"
    },
    {
      "id": 10000000000000000056,
      "name": "Example Processor",
      "project": "Example project",
      "code": "function(a) { return a+'.'+a; }"
    },
    {
      "id": 10000000000000000057,
      "name": "Example Processor",
      "project": "Example project",
      "code": "function(a) { return a+'.'+a; }"
    }
  ]
}
```

POST /rest/api/1/processor/<id>

Updates a processor.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the processor to update
name	String	a speaking name to identify the processor
project	String	a reference (ID) to the Project
code	String	JavaScript code of the processor

Responses

Status	Description
404	If no entity with the given ID exists.
400	If the request parameter contain errors.
200	The new created entity is returned

Example 1

```
curl --request POST --data "name=Example processor&project=9998&code=function() {}" http://localhost:9000/rest/api/1/processor/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Processor with id 9999."
```

Example 2

```
curl --request POST --data "name=Example processor 2&project=9898&code=function() {}" http://localhost:9000/rest/api/1/processor/10000000000000000055
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"project":["Invalid value"]}
```

Example 3

```
curl --request POST --data "name=Example processor&project=10000000000000000043&code=function(a) { return a+'.'+a; }" http://localhost:9000/rest/api/1/processor/10000000000000000056
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{
  "id": 10000000000000000056,
  "name": "Example processor",
  "project": "The Example Project",
  "code": "function(a) { return a+'.'+a; }"
}
```

GET /rest/api/1/processor/<id>

Returns one processor.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to get

Responses

Status	Description
404	If no entity with the given ID exists.
200	If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/processor/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Processor with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/processor/1000000000000000054
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{ "id": 1000000000000000054, "name": "Example Processor", "project":
{ "id": 3315, "name": "Example project", "code": "function(a) { return a+'.'+a; }" }
```

DELETE /rest/api/1/processor/<id>

Deletes a processor.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to delete

Responses

Status	Description
404	If the processor to delete could not be found.
204	If the processor was deleted.

Example 1

```
curl --request DELETE http://localhost:9000/rest/api/1/processor/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Processor with id 9999."
```

Example 2

```
curl --request DELETE http://localhost:9000/rest/api/1/processor
/1000000000000000057
```

The previous command **did** return with status code **204 (No Content)** and yielded the following:

3.4 Project Planning Tool**GET /rest/api/1/ppt**

Returns all Project Planning Tools.

[Authentication](#) is required to use this method.

Responses

Status	Description
200	It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/ppt
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{ "items": [ { "id": 3321, "name": "Example PPT" }, { "id": 3324, "name": "Example PPT" },
{ "id": 3327, "name": "Example PPT" }, { "id": 3330, "name": "Example PPT" },
{ "id": 3348, "name": "Example Project Planning Tool" }, { "id": 3350, "name": "Example
Project Planning Tool" }, { "id": 100000000000000005, "name": "Example Jira" },
{ "id": 1000000000000000041, "name": "Example Jira" },
{ "id": 1000000000000000051, "name": "Example Jira" } ] }
```

GET /rest/api/1/ppt/<id>

Returns one Project Planning Tool.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
------	------	-------------

id	Long	The id of the entity to get
----	------	-----------------------------

Responses

Status	Description
404	If no entity with the given ID exists.
200	If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/ppt/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Project Planning Tool with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/ppt/10000000000000000051
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":10000000000000000051,"name":"Example Jira"}
```

POST /rest/api/1/ppt/createPPTTask

Creates a Task on a remote Project Planning Tool Server and stores the creation on the server.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
account	Long	The id of an EEPPI-account of the currently logged in user.
path	String	The path on the remote server beginning with a '/'
content	JsonNode	Json-data to be sent to the remote server
taskTemplate	Long	The id of a TaskTemplate of which this Task is created.

taskProperties[]	String	This parameter can be passed multiple times. It represents a Task Property Value for this Task. The Format is "{ID of the Task Property}-{The Value of it}".
project	Long	The id of a Project in which this Task is created.

Responses

Status	Description
400	If there is an error during preparation of the request for the remote server.
502	If the remote server could not be found.
504	If the remote server did not respond.
-	The return value from the remote server is returned (the Json if it's a Json or a Json containing the type and the content as a simple Json-Object).

Example 1

```
curl --request POST --data "account=100&path=/rest/api/2/issue/&content={
  "fields": {
    "project":
      {
        "key": "PRV"
      }
    },
    "summary": "My generated issue",
    "description": "This is an issue, which is created by EEPPI over the
API",
    "issuetype": {
      "name": "Task"
    }
  }
}&taskTemplate=51&taskProperties[]=53-Example Value&project=55"
http://localhost:9000/rest/api/1/ppt/createPPTTask
```

The previous command **would probably** return with status code **201** and yield the following:

```
{
  "id": "10000",
  "key": "PRV-24",
  "self": "http://jira.example.ch/jira/rest/api/2/issue/10000"
}
```

Example 2

```
curl --request POST --data "account=100&path=/index.html&content=
{}&taskTemplate=51&taskProperties[]=53-Example Value&project=55"
http://localhost:9000/rest/api/1/ppt/createPPTTask
```

The previous command **would probably** return with status code **200** and yield the following:

```
{
  "content": "<html><head></head><body>...</body></html>",
  "type": "text/html; charset=utf-8"
}
```

Example 3

```
curl --request POST --data "account=not a number&path=not a path&content=no
Json&taskTemplate=wrongFormat&taskProperties[]=9999" http://localhost:9000
/rest/api/1/ppt/createPPTTask
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"taskProperties[0]":["Invalid value"],"project":["This field is
required"],"taskTemplate":["Invalid value"],"content":["Invalid
value"],"account":["Invalid value"]}
```

3.5 Request Template

POST /rest/api/1/requestTemplate

Stores a new Request Template for sending a Task to a Project Planning Tool.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
name	String	the new name of the new Request Template
ppt	String	a reference (ID) to the Project Planning Tool
project	String	a reference (ID) to the Project
url	String	the URL to call on the Project Planning Tool (only the part after the domain and port, beginning with a "/")
requestTemplate	String	the template for the request to be performed (including markup for usages of Processors)

Responses

Status	Description
400	If the Request Template could not be stored.
200	If the Request Template was stored. It is also returned in Json form.

Example 1

```
curl --request POST --data "name=Request Template name&ppt=9999&project=9998&
url=/some/target&requestTemplate={}" http://localhost:9000/rest/api
/1/requestTemplate
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"ppt":["Invalid value"],"project":["Invalid value"]}
```

Example 2

```
curl --request POST --data "name=Request Template name&ppt=100000000000000041&
project=100000000000000042&url=/example/target&requestTemplate={}"
http://localhost:9000/rest/api/1/requestTemplate
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":3355,"ppt":{"id":100000000000000041,"name":"Example Jira"},"project":
{"id":100000000000000042,"name":"The Example Project"},"name":"Request
Template name","url":"/example/target","requestBodyTemplate":null}
```

GET /rest/api/1/requestTemplate

Returns all Request Templates for sending a Task to a Project Planning Tool.

[Authentication](#) is required to use this method.

Responses

Status	Description
200	It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/requestTemplate
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"items":[{"id":3355,"ppt":{"id":100000000000000041,"name":"Example
Jira"},"project":{"id":100000000000000042,"name":"The Example
Project"},"name":"Request Template name","url":"/example
/target","requestBodyTemplate":null},{"id":100000000000000045,"ppt":
{"id":3327,"name":"Example PPT"},"project":{"id":3328,"name":"Example
Project"},"name":"My Request Template","url":"/example
/endpoint","requestBodyTemplate":{"name":"${title}"},
{"id":100000000000000047,"ppt":{"id":3321,"name":"Example PPT"},"project":
{"id":3322,"name":"Example Project"},"name":"My Request
Template","url":"/example/endpoint","requestBodyTemplate":{"name":"${title}"}
```

```
{\}, {"id":1000000000000000048, "ppt": {"id":3330, "name": "Example PPT"}, "project": {"id":3331, "name": "Example Project"}, "name": "My Request Template", "url": "/example/endpoint", "requestBodyTemplate": "{\name\":"\${title}\"}"}, {"id":1000000000000000049, "ppt": {"id":3324, "name": "Example PPT"}, "project": {"id":3325, "name": "Example Project"}, "name": "My Request Template", "url": "/example/endpoint", "requestBodyTemplate": "{\name\":"\${title}\"}"}]}
```

POST /rest/api/1/requestTemplate/<id>

Updates a Request Template for sending a Task to a Project Planning Tool.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the Request Template to update
name	String	the new name of the Request Template to update
ppt	String	a reference (ID) to the Project Planning Tool
project	String	a reference (ID) to the Project
url	String	the URL to call on the Project Planning Tool (only the part after the domain and port, beginning with a "/")
requestTemplate	String	the template for the request to be performed (including markup for usages of Processors)

Responses

Status	Description
404	If no entity with the given ID exists.
400	If the request parameter contain errors.
200	The new created entity is returned

Example 1

```
curl --request POST --data "name=Request Template name&ppt=9999&project=9998&url=/asdf&requestTemplate={}" http://localhost:9000/rest/api/1/requestTemplate/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Request Template with id 9999."
```

Example 2

```
curl --request POST --data "name=Request Template name&ppt=9988&project=9977&url=/example/target&requestTemplate={}" http://localhost:9000/rest/api/1/requestTemplate/1000000000000000047
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"ppt":["Invalid value"],"project":["Invalid value"]}
```

Example 3

```
curl --request POST --data "name=Request Template name&ppt=1000000000000000041&project=1000000000000000042&url=/example/target&requestTemplate={}" http://localhost:9000/rest/api/1/requestTemplate/1000000000000000049
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":1000000000000000049, "ppt": {"id":1000000000000000041, "name": "Example Jira"}, "project": {"id":1000000000000000042, "name": "The Example Project"}, "name": "Request Template name", "url": "/example/target", "requestBodyTemplate": null}
```

GET /rest/api/1/requestTemplate/<id>

Returns one Request Template for sending a Task to a Project Planning Tool.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to get

Responses

Status	Description
404	If no entity with the given ID exists.
200	If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/requestTemplate/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Request Template with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/requestTemplate/1000000000000000045
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":1000000000000000045,"ppt":{"id":3327,"name":"Example PPT"},"project":{"id":3328,"name":"Example Project"},"name":"My Request Template","url":"/example/endpoint","requestBodyTemplate":{"name":"${title}"}}
```

DELETE /rest/api/1/requestTemplate/<id>

Deletes a Request Template for sending a Task to a Project Planning Tool.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to delete

Responses

Status	Description
404	If the Request Template to delete could not be found.
204	If the Request Template was deleted.

Example 1

```
curl --request DELETE http://localhost:9000/rest/api/1/requestTemplate/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Request Template with id 9999."
```

Example 2

```
curl --request DELETE http://localhost:9000/rest/api/1/requestTemplate/1000000000000000048
```

The previous command **did** return with status code **204 (No Content)** and yielded the following:

3.6 Task Property

POST /rest/api/1/taskProperty

Creates a new Task Property.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
name	String	The name of the new Task Property

Responses

Status	Description
400	If the request parameter contain errors.
200	The new created entity is returned.

Example 1

```
curl --request POST --data "name=A new Task Property" http://localhost:9000/rest/api/1/taskProperty
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":3356,"name":"A new Task Property"}
```

GET /rest/api/1/taskProperty

Reads all Task Properties.

[Authentication](#) is required to use this method.

Responses

Status	Description
200	It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/taskProperty
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{
  "items": [
    { "id": 3340, "name": "My example Task Property 2" },
    { "id": 3343, "name": "My example Task Property 2" },
    { "id": 3356, "name": "A new Task Property" },
    { "id": 1000000000000000008, "name": "My example Task Property" },
    { "id": 1000000000000000013, "name": "My example Task Property" },
    { "id": 1000000000000000023, "name": "My example Task Property" },
    { "id": 1000000000000000027, "name": "My example Task Property" }
  ]
}
```

POST /rest/api/1/taskProperty/<id>

Updates an existing Task Property with new data.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the Task Property to update
name	String	The new name of the Task Property

Responses

Status	Description
404	If no entity with the given ID exists.
400	If the request parameter contain errors.
200	The new created entity is returned

Example 1

```
curl --request POST --data "name=My beautiful task property"
http://localhost:9000/rest/api/1/taskProperty/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the

following:

```
"Could not find Task Property with id 9999."
```

Example 2

```
curl --request POST --data "name=My example Task Property" http://localhost:9000
/rest/api/1/taskProperty/1000000000000000008
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{ "id": 1000000000000000008, "name": "My example Task Property" }
```

GET /rest/api/1/taskProperty/<id>

Reads a Task Property.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to get

Responses

Status	Description
404	If no entity with the given ID exists.
200	If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/taskProperty/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Property with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/taskProperty/1000000000000000013
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{ "id":10000000000000000013, "name": "My example Task Property" }
```

DELETE /rest/api/1/taskProperty/<id>

Deletes a Task Property.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to delete

Responses

Status	Description
404	If no entity with the given ID exists.
409	If the entity could not be deleted.
204	If the entity is successfully deleted.

Example 1

```
curl --request DELETE http://localhost:9000/rest/api/1/taskProperty/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Property with id 9999."
```

Example 2

```
curl --request DELETE http://localhost:9000/rest/api/1/taskProperty/10000000000000000023
```

The previous command **did** return with status code **204 (No Content)** and yielded the following:

3.7 Task Template

POST /rest/api/1/taskTemplate

Creates a new Task Template of which (concrete) Tasks then can be generated.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
name	String	The name of the new Task Template

Responses

Status	Description
400	If the request parameter contain errors.
200	The new created entity is returned.

Example 1

```
curl --request POST --data "name=A new Task Template" http://localhost:9000/rest/api/1/taskTemplate
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{ "id":3357, "properties": [], "parent": null, "name": "A new Task Template" }
```

GET /rest/api/1/taskTemplate

Reads all Task Templates.

[Authentication](#) is required to use this method.

Responses

Status	Description
200	It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/taskTemplate
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{ "items": [ { "id":3357, "properties": [], "parent": null, "name": "A new Task
```

```

Template"}, {"id":10000000000000000008,"properties":[],"parent":null,"name":"My
example Task Template"}, {"id":10000000000000000013,"properties":
[], "parent":null,"name":"My example Task Template"},
{"id":10000000000000000025,"properties":[],"parent":null,"name":"My example Task
Template"}, {"id":10000000000000000031,"properties":[],"parent":null,"name":"My
example Task Template"}, {"id":10000000000000000033,"properties":
[], "parent":null,"name":"My example Task Template"}, {"id":3301,"properties":
[], "parent":null,"name":"My example Task Template 2"}, {"id":3303,"properties":
[], "parent":null,"name":"My example Task Template 2"}, {"id":3305,"properties":
[], "parent":null,"name":"My example Task Template 2"}, {"id":3341,"properties":
[{"id":10000000000000000029,"property":{"id":3340,"name":"My example Task
Property 2"},"value":"My example Value"}], "parent":null,"name":"My example Task
Template 2"}, {"id":3344,"properties":[{"id":10000000000000000030,"property":
{"id":3343,"name":"My example Task Property 2"},"value":"My example
Value"}], "parent":null,"name":"My example Task Template
2"}, {"id":3307,"properties":[],"parent":null,"name":"My example Task Template
7"}]}

```

POST /rest/api/1/taskTemplate/<id>

Updates an existing Task Template with new data.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the Task Template to update
name	String	The new name of the Task Template

Responses

Status	Description
404	If no entity with the given ID exists.
400	If the request parameter contain errors.
200	The new created entity is returned

Example 1

```
curl --request POST --data "name=My beautiful task template"
http://localhost:9000/rest/api/1/taskTemplate/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Template with id 9999."
```

Example 2

```
curl --request POST --data "name=My example Task Template" http://localhost:9000
/rest/api/1/taskTemplate/10000000000000000008
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":10000000000000000008,"properties":[],"parent":null,"name":"My example Task
Template"}
```

GET /rest/api/1/taskTemplate/<id>

Reads a Task Template.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to get

Responses

Status	Description
404	If no entity with the given ID exists.
200	If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/taskTemplate/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Template with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/taskTemplate/10000000000000000008
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":10000000000000000008,"properties":[],"parent":null,"name":"My example Task
Template"}
```


DELETE /rest/api/1/taskTemplate/<id>

Deletes a Task Template.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to delete

Responses

Status	Description
404	If no entity with the given ID exists.
409	If the entity could not be deleted.
204	If the entity is successfully deleted.

Example 1

```
curl --request DELETE http://localhost:9000/rest/api/1/taskTemplate/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Template with id 9999."
```

Example 2

```
curl --request DELETE http://localhost:9000/rest/api/1/taskTemplate/10000000000000000013
```

The previous command **did** return with status code **204 (No Content)** and yielded the following:

POST /rest/api/1/taskTemplate/<id>/addProperty

Adds a new property to an existing Task Template.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the Task Template
property	String	The id of the Task Property
value	String	The value of the Property

Responses

Status	Description
404	If no Task Template with the given ID exists
400	If the request parameters contain errors.
200	The Task Template containing the new Property is returned

Example 1

```
curl --request POST --data "property=8888&value=My beautiful task value" http://localhost:9000/rest/api/1/taskTemplate/9999/addProperty
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Template with id 9999."
```

Example 2

```
curl --request POST --data "property=10000000000000000027&value=My example Task Value" http://localhost:9000/rest/api/1/taskTemplate/10000000000000000025/addProperty
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":10000000000000000025,"properties":[{"id":3358,"property":{"id":10000000000000000027,"name":"My example Task Property"},"value":"My example Task Value"}],"parent":null,"name":"My example Task Template"}
```

POST /rest/api/1/taskTemplate/<id>/properties/<taskTemplate>

Updates a task property value.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the Task Template Value
taskTemplate	Long	The id of the Task Template
property	String	The id of the Task Property
value	String	The value of the Property

Responses

Status	Description
404	If no Task Template or Task Property Value with the given ID exists
400	If the request parameters contain errors.
200	The Task Template containing the updated Property is returned

Example 1

```
curl --request POST --data "property=8888&value=My beautiful task template"
http://localhost:9000/rest/api/1/taskTemplate/9999/properties/7777
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Template with id 7777."
```

Example 2

```
curl --request POST --data "property=1000000000000000027&value=My example
Value" http://localhost:9000/rest/api/1/taskTemplate/1000000000000000029
/properties/1000000000000000025
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Template with id 1000000000000000025."
```

DELETE /rest/api/1/taskTemplate/<id>/properties/<taskTemplate>

Deletes a task property value.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the Task Template Value
taskTemplate	Long	The id of the Task Template

Responses

Status	Description
404	If no Task Template or Task Property Value with the given ID exists
200	The Task Template containing the removed Property is returned

Example 1

```
curl --request DELETE http://localhost:9000/rest/api/1/taskTemplate
/9999/properties/7777
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Template with id 7777."
```

Example 2

```
curl --request DELETE http://localhost:9000/rest/api/1/taskTemplate
/10000000000000000030/properties/1000000000000000031
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Task Template with id 1000000000000000031."
```

3.8 PPTAccount**POST /rest/api/1/user/pptAccount**

Stores a new login information for a Project Planning Tool.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
------	------	-------------

ppt	String	a reference (ID) to the Project Planning Tool
urlUrl	String	the URL to the Project Planning Tool
pptUsername	String	the username for the user for the Project Planning Tool
pptPassword	String	the password for the user for the Project Planning Tool

Responses

Status	Description
400	If the login information could not be stored.
200	If the login information were stored. They are also returned in Json form.

Example 1

```
curl --request POST --data "ppt=9999&urlUrl=no url&pptUsername=name&pptPassword=1234" http://localhost:9000/rest/api/1/user/pptAccount
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"pptUrl":{"This field is required"},"ppt":{"Invalid value"}}
```

Example 2

```
curl --request POST --data "ppt=1000000000000000005&urlUrl=http.jira.example.com&pptUsername=admin&pptPassword=12345678" http://localhost:9000/rest/api/1/user/pptAccount
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"pptUrl":{"This field is required"}}
```

GET /rest/api/1/user/pptAccount

Returns all login information (but the password) for the currently logged in user for Project Planning Tools.

[Authentication](#) is required to use this method.

Responses

Status	Description
--------	-------------

200 It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/user/pptAccount
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"items":[{"id":1000000000000000003,"ppt":{"id":3348,"name":"Example Project Planning Tool"},"pptUrl":"https://ppt.example.com","user":{"id":3295,"name":"user0"},"pptUsername":"tbucher"}, {"id":1000000000000000007,"ppt":{"id":3350,"name":"Example Project Planning Tool"},"pptUrl":"https://ppt.example.com","user":3295,"pptUsername":"tbucher"}]}
```

POST /rest/api/1/user/pptAccount/<id>

Updates login information for a Project Planning Tool on the server.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the login information to update
ppt	String	a reference (ID) to the Project Planning Tool
pptUrl	String	the URL to the Project Planning Tool
pptUsername	String	the username for the user for the Project Planning Tool
pptPassword	String	the password for the user for the Project Planning Tool (optional)

Responses

Status	Description
404	If no entity with the given ID exists.
400	If the request parameter contain errors.
200	The new created entity is returned

Example 1

```
curl --request POST --data "ppt=1&pptUrl=no url&pptUsername=name&pptPassword=1234" http://localhost:9000/rest/api/1/user/pptAccount/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Projectplanningtool Account with id 9999."
```

Example 2

```
curl --request POST --data "ppt=9999&pptUrl=no url&pptUsername=ozander&pptPassword=pMuE2ekiDa" http://localhost:9000/rest/api/1/user/pptAccount/10000000000000000003
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"ppt":{"Invalid value"}}
```

Example 3

```
curl --request POST --data "ppt=1&pptUrl=https://ppt.example.com&pptUsername=tbucher&pptPassword=7YqupNxN9v" http://localhost:9000/rest/api/1/user/pptAccount/10000000000000000003
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"ppt":{"Invalid value"}}
```

GET /rest/api/1/user/pptAccount/<id>

Returns one login information (but the password) for the currently logged in user for Project Planning Tools.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to get

Responses

Status	Description
404	If no entity with the given ID exists.

200 If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/user/pptAccount/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Projectplanningtool Account with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/user/pptAccount/10000000000000000003
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":10000000000000000003,"ppt":{"id":3348,"name":"Example Project Planning Tool"},"pptUrl":"https://ppt.example.com","user":{"id":3295,"name":"user0"},"pptUsername":"tbucher"}
```

DELETE /rest/api/1/user/pptAccount/<id>

Deletes login information for a Project Planning Tool on the server.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to delete

Responses

Status	Description
404	If the login information to delete could not be found.
204	If the login information were deleted.

Example 1

```
curl --request DELETE http://localhost:9000/rest/api/1/user/pptAccount/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the

following:

```
"Could not find Projectplanningtool Account with id 9999."
```

Example 2

```
curl --request DELETE http://localhost:9000/rest/api/1/user/pptAccount/10000000000000000007
```

The previous command **did** return with status code **204 (No Content)** and yielded the following:

3.9 Project

GET /rest/api/1/project

Returns all Projects.

[Authentication](#) is required to use this method.

Responses

Status	Description
200	It's always a list returned containing all (but if there is none also none) entities.

Example 1

```
curl http://localhost:9000/rest/api/1/project
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{
  "items": [
    { "id": 3310, "name": "Example project" },
    { "id": 3312, "name": "Example project" },
    { "id": 3315, "name": "Example project" },
    { "id": 3317, "name": "Example project" },
    { "id": 3322, "name": "Example Project" },
    { "id": 3325, "name": "Example Project" },
    { "id": 3328, "name": "Example Project" },
    { "id": 3331, "name": "Example Project" },
    { "id": 10000000000000000042, "name": "The Example Project" },
    { "id": 10000000000000000043, "name": "The Example Project" },
    { "id": 10000000000000000059, "name": "The Example Project" }
  ]
}
```

GET /rest/api/1/project/<id>

Returns one Project.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
id	Long	The id of the entity to get

Responses

Status	Description
404	If no entity with the given ID exists.
200	If it's found, it's returned.

Example 1

```
curl http://localhost:9000/rest/api/1/project/9999
```

The previous command **did** return with status code **404 (Not Found)** and yielded the following:

```
"Could not find Project with id 9999."
```

Example 2

```
curl http://localhost:9000/rest/api/1/project/10000000000000000059
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":10000000000000000059,"name":"The Example Project"}
```

3.10 User

POST /rest/api/1/user/changePassword

This changes the password of an EEPPI-user.

[Authentication](#) is required to use this method.

Parameters

Name	Type	Description
oldPassword	String	the current password for the user
newPassword	String	the new password for the user

newPasswordRepeat String the new password for the user (repetition, to guarantee the user didn't make a typo)

Responses

Status	Description
400	If the password could not be changed.
200	If the password was changed.

Example 1

```
curl --request POST --data "oldPassword=demo&newPassword=1234&newPasswordRepeat=1234" http://localhost:9000/rest/api/1/user/changePassword
```

The previous command **would probably** return with status code **200** and yield the following:

Example 2

```
curl --request POST --data "oldPassword=demo&newPassword=1234&newPasswordRepeat=another password" http://localhost:9000/rest/api/1/user/changePassword
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"":["The two passwords do not match"]}
```

POST /rest/api/1/user/login

Checks the login information for the user and if the login is successful a cookie is set.

This method is public and can be used without authentication.

Parameters

Name	Type	Description
name	String	username
password	String	the password for the user

Responses

Status	Description
400	If the user could not be logged in.
200	If the user could be logged in, and a cookie is set.

Example 1

```
curl --request POST --data "name=demo&password=demo" http://localhost:9000/rest/api/1/user/login
```

The previous command **would probably** return with status code **200** and yield the following:

```
{"id":1,"name":"demo"}
```

Example 2

```
curl --request POST --data "name=demo&password=invalid password" http://localhost:9000/rest/api/1/user/login
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
Username or Password wrong
```

Example 3

```
curl --request POST --data "name=demo&password=" http://localhost:9000/rest/api/1/user/login
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:

```
{"password":["This field is required"]}
```

GET /rest/api/1/user/loginStatus

Returns the login status for the currently logged in user and a Json representation of it.

This method is public and can be used without authentication.

Responses

Status	Description
200	Is always returned, and a Json containing either nothing (if no user is logged in) or the user.

Example 1

```
curl http://localhost:9000/rest/api/1/user/loginStatus
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":3295,"name":"user0"}
```

Example 2

```
curl http://localhost:9000/rest/api/1/user/loginStatus
```

The previous command **would probably** return with status code **200** and yield the following:

```
{
  "id": 1,
  "name": "demo"
}
```

POST /rest/api/1/user/logout

Does log out the currently logged in user by removing the cookie.

This method is public and can be used without authentication.

Responses

Status	Description
204	Is always returned, and the login cookie is being removed.

Example 1

```
curl --request POST http://localhost:9000/rest/api/1/user/logout
```

The previous command **did** return with status code **204 (No Content)** and yielded the following:

POST /rest/api/1/user/register

This creates a new EEPPI-user.

This method is public and can be used without authentication.

Parameters

Name	Type	Description
name	String	username
password	String	the new password for the user
passwordRepeat	String	the new password for the user (repetition, to guarantee the user didn't make a typo)

Responses

Status	Description
400	If the user could not be created.
200	If the user could be created, it is returned.

Example 1

```
curl --request POST --data "name=New Username 1&password=1234&passwordRepeat=1234" http://localhost:9000/rest/api/1/user/register
```

The previous command **did** return with status code **200 (OK)** and yielded the following:

```
{"id":3359,"name":"New Username 1"}
```



Example 2

```
curl --request POST --data "name=New Username 2&password=1234&passwordRepeat=another password" http://localhost:9000/rest/api/1/user/register
```

The previous command **did** return with status code **400 (Bad Request)** and yielded the following:


```
{"":["The two passwords do not match"]}
```

E.2. Client Code


Index

Modules

-  [app](#)

Modules


app

 [app](#): *app*

Defined in [PersistentEntity.ts:1](#)
 Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:2](#)
 Defined in [Repository.ts:4](#)

Index

Modules

-  [domain](#)

Modules

domain

⚙️ ☰

Defined in [PersistentEntity.ts:1](#)
 Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:2](#)
 Defined in [Repository.ts:4](#)

Index

Modules

- 🏠 factory
- 🏠 repository

Modules

factory

🏠 factory: [factory](#)

Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:2](#)

Index

Classes

- 🏠 [Empty](#)
- 🏠 [ObjectFactory](#)

Classes

⚙️ ☰

🏠 [Empty](#): [Empty](#)

Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:3](#)

ObjectFactory

🏠 [ObjectFactory](#): [ObjectFactory](#)

Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:5](#)

static [createFromJson](#)

🏠 [createFromJson](#)(type: *any*, data: *any*): *any*

Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:15](#)

create an prototype-object from object data recursive

Parameters

- **type:** *any*
e.g. String, Number, Object, Array, YourPrototype, ... -> Own Prototypes must implement static property factoryConfiguration (core.FactoryConfiguration)
- **data:** *any*
e.g. { "id": 5, "name": "DummyObject" }

Returns *any*

- New object

static [createObject](#)

🏠 [createObject](#)(constructor: *any*, args: *any*): *any*

Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:90](#)

properties will fail on access, so use an empty dummy class

Parameters

- **constructor:** *any*
- **args:** *any*

Returns *any*

static createProperty

`createProperty(dataItem: any, property: any): any`

Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:64](#)

create an object property from a data item

Parameters

- **dataItem:** *any*
e.g. "DummyObject1"
- **property:** *any*
e.g. { name: "id", type: Number, subType: null }

Returns *any*

static updateFromJson

`updateFromJson(item: any, type: any, data: any): any`

Defined in [/home/tobias/Business/studium/modules/BA/HSR.BA.Project/app/assets/scripts/classes/domain/factory/ObjectFactory.ts:45](#)

updates an object with some new data

Parameters

- **item:** *any*
the item to be updated
- **type:** *any*
e.g. String, Number, Object, Array, YourPrototype, ... -> Own

- **data:** *any*
e.g. { "id": 5, "name": "DummyObject" }

Returns *any*

- Updated object

repository

`repository: repository`

Defined in [PersistentEntity.ts:1](#)
Defined in [Repository.ts:4](#)

Index

Modules

`core`

Modules

core

`core: core`

Defined in [PersistentEntity.ts:1](#)
Defined in [Repository.ts:4](#)

Index

⚙️ ☰

📦 **PersistentEntity**

Classes

📦 [Repository](#)

Interfaces

PersistentEntity

📦 PersistentEntity: [PersistentEntity](#)

Defined in [PersistentEntity.ts:2](#)

id

○ id: *number*

Defined in [PersistentEntity.ts:3](#)

Classes

Repository

📦 Repository: [Repository](#)

Defined in [Repository.ts:5](#)

constructor

⦿ constructor(httpService: *any*): [Repository](#)

Defined in [Repository.ts:67](#)

⚙️ ☰

- [httpService: any](#)
 - Angular \$http service, used to call a remote api

Returns [Repository](#)

dataList

● dataList: *string*

Defined in [Repository.ts:35](#)

Name of the item list inside the returned js object from remote api

example `// api returns:`

```

{
  "items": [
    {
      "name": "item 1"
    },
    {
      "name": "item 2"
    }
  ]
}

```

filter

● filter: *Function*

Defined in [Repository.ts:46](#)

Function to filter the items returned by the remote api

example `function(element) {`

```

return element.type

=== "domain.model.Asset";

```

⚙️ ☰

}

filter function signature

`() (element: any): boolean`

Defined in [Repository.ts:46](#)

Parameters

- **element:** *any*
 - The current item inside the items list

Returns *boolean*

host

- **host:** *string*

Defined in [Repository.ts:16](#)

The host url of the remote api, will be concatenated in front of the relative resource path. Should be used in combination of proxy or host has to allow cross origin calls.

example `"http://dks.eeppi.ch"`

httpClient

- **httpClient:** *any*

Defined in [Repository.ts:48](#)

itemCache

⚙️ ☰

```

approximateRepositoryRepositorySystemEntry:
  
```

Defined in [Repository.ts:8](#)

proxy

- **proxy:** *Object*

Defined in [Repository.ts:24](#)

Path with {target} to be replaced by the remote url. If proxy is set, /path/to/proxy/{[http://example.host.tld/path/to/resource/](#)} will be called instead of [http://example.host.tld/path/to/resource/](#)

example `"/dks/getFromDKS?url={target}"`

proxy.url

- **proxy.url:** *string*

Defined in [Repository.ts:24](#)

resources

- **resources:** *Object*

Defined in [Repository.ts:62](#)

Dictionary with resource paths

example `{`

```

    list: {
      method: 'GET', url: '/dksMapping' },
    detail: { method:
'GET', url: '/dksMapping/{id}' },
  
```

```
'POST', url: '/dksMapping' },

      update: { method:
'POST', url: '/dksMapping/{id}' },

      remove: { method:
'POST', url: '/dksMapping/{id}/delete' }

    }
  }
}
```

resources[]

```
[] (index: string): { method: string; url: string; }
```

Defined in [Repository.ts:62](#)

Parameters

- **index:** *string*

Returns *{ method: string; url: string; }*

type

- type: *any*

Defined in [Repository.ts:7](#)

add

```
() add(item: T in app.domain.repository.core.Repository<T extends PersistentEntity>, callback: Function)
```

Defined in [Repository.ts:136](#)

Parameters

- **item:** *T in app.domain.repository.core.Repository<T extends PersistentEntity>*
 - The item to persist using the remote api
- **callback:** *Function*
 - The callback is called with (true, item) on success and with (false, null) on error

callback function signature

```
() (success: boolean, item: T in app.domain.repository.core.Repository<T extends PersistentEntity>)
```

Defined in [Repository.ts:136](#)

Parameters

- **success:** *boolean*
- **item:** *T in app.domain.repository.core.Repository<T extends PersistentEntity>*

findAll

```
() findAll(callback: Function, doCache?: boolean = false)
```

Defined in [Repository.ts:92](#)

Find all items in remote repository or local cache.

Parameters

- **callback:** *Function*
 - Will be called with (true, items) on success successful remote/cache call and with (false, []) on error

callback function signature

```
() (success: boolean, items: Array<T extends app.domain.repository.core.PersistentEntity>)
```

Defined in [Repository.ts:92](#)

- **success:** *boolean*
- **items:** *Array<T extends app.domain.repository.core.PersistentEntity>*
- **doCache?:** *boolean* optional
 - Returns items from remote api and updates the items cache if false, returns items from local cache if true.

findAllWithNodesAndSubNodes

```
() findAllWithNodesAndSubNodes(propertyName: string, repository: Repository, callback: Function, doCache?: boolean = false)
```

Defined in [Repository.ts:315](#)

Find all items and its related properties

Parameters

- **propertyName:** *string*
 - The name of the item property
- **repository:** *Repository*
 - The repository to fetch the the property
- **callback:** *Function*
 - Will be called with (true, items) on success successful remote/cache call and with (false, []) on error
- **callback function signature**

```
() (success: boolean, items: Array<T extends app.domain.repository.core.PersistentEntity>)
```

Defined in [Repository.ts:316](#)

Parameters

- **success:** *boolean*
- **items:** *Array<T extends app.domain.repository.core.PersistentEntity>*
- **doCache?:** *boolean* optional
 - Returns items from remote api and updates the item cache if

findByPropertyId

```
() findByPropertyId(propertyName: string, property: any, callback: Function, doCache?: boolean = false)
```

Defined in [Repository.ts:244](#)

Find items by a property (object) id

Parameters

- **propertyName:** *string*
 - The name of the property of the item
- **property:** *any*
 - The value of the property to search
- **callback:** *Function*
 - The callback is called with (true, item) on success and with (false, null) on error
- **callback function signature**

```
() (success: boolean, items: Array<T extends app.domain.repository.core.PersistentEntity>)
```

Defined in [Repository.ts:244](#)

Parameters

- **success:** *boolean*
- **items:** *Array<T extends app.domain.repository.core.PersistentEntity>*
- **doCache?:** *boolean* optional
 - Returns item from remote api and updates the item in the cache if false, returns item from local cache if true.

findOneBy

```
() findOneBy(propertyName: string, property: any, callback:
```

Defined in [Repository.ts:204](#)

Find an item by a property value

Parameters

- **propertyName:** *string*
 - The name of the property of the item
- **property:** *any*
 - The value of the property to search
- **callback:** *Function*
 - The callback is called with (true, item) on success and with (false, null) on error

callback function signature

```
() (success: boolean, item: T in app.domain.repository.core.Repository<T extends PersistentEntity>)
```

Defined in [Repository.ts:204](#)

Parameters

- **success:** *boolean*
- **item:** *T* in *app.domain.repository.core.Repository<T extends PersistentEntity>*
- **doCache?:** *boolean* optional
 - Returns item from remote api and updates the item in the cache if false, returns item from local cache if true.

findOneById

```
() findOneById(id: number, callback: Function, doCache?: boolean = false)
```

Defined in [Repository.ts:164](#)

Search item by id on remote resource or in cache. Updates item in item cache on success.

Parameters

- **callback:** *Function*
 - The callback is called with (true, item) on success and with (false, null) on error

callback function signature

```
() (success: boolean, item: T in app.domain.repository.core.Repository<T extends PersistentEntity>)
```

Defined in [Repository.ts:164](#)

Parameters

- **success:** *boolean*
- **item:** *T* in *app.domain.repository.core.Repository<T extends PersistentEntity>*
- **doCache?:** *boolean* optional
 - Returns item from remote api and updates the item in the cache if false, returns item from local cache if true.

findOneByPropertyId

```
() findOneByPropertyId(propertyName: string, property: any, callback: Function, doCache?: boolean = false)
```

Defined in [Repository.ts:224](#)

Find an item by a property (object) id

Parameters

- **propertyName:** *string*
 - The name of the property of the item
- **property:** *any*
 - The value of the property to search
- **callback:** *Function*
 - The callback is called with (true, item) on success and with (false, null) on error
- **callback function signature**

`app.domain.repository.core.Repository<T> extends PersistentEntity>`

Defined in [Repository.ts:224](#)

Parameters

- **success:** *boolean*
- **item:** *T in app.domain.repository.core.Repository<T extends PersistentEntity>*
- **doCache?:** *boolean* optional
 - Returns item from remote api and updates the item in the cache if false, returns item from local cache if true.

findSubNodes

```
() findSubNodes(nodes: Array<T extends app.domain.repository.core.PersistentEntity>,
  propertyName: string, repository: Repository, callback:
  Function, doCache?: boolean = false)
```

Defined in [Repository.ts:337](#)

Find related properties of nodes

Parameters

- **nodes:** *Array<T extends app.domain.repository.core.PersistentEntity>*
 - A list with items to find its properties
- **propertyName:** *string*
 - The name of the item property
- **repository:** *Repository*
 - The repository to fetch the the property
- **callback:** *Function*
 - Will be called with (true, items) on success successful remote/cache call and with (false, []) on error
- **callback function signature**

```
() (success: boolean, items: Array<T extends
```

Defined in [Repository.ts:341](#)

Parameters

- **success:** *boolean*
- **items:** *Array<T extends app.domain.repository.core.PersistentEntity>*
- **doCache?:** *boolean* optional
 - Returns items from remote api and updates the item cache if false, returns items from local cache if true.

getResourcePath

```
() getResourcePath(resource: string): string
```

Defined in [Repository.ts:78](#)

Get url for resource

Parameters

- **resource:** *string*

Returns

string
path - Gets the path for the requested resource. Concatenates host & path and uses proxy of set

remove

```
() remove(item: T in app.domain.repository.core.Repository<T extends PersistentEntity>, callback: Function)
```

Defined in [Repository.ts:262](#)

Remove item from remote collection and item cache

Parameters

- **item:** *T in app.domain.repository.core.Repository<T extends PersistentEntity>*
 - The item to remove

error

- **callback function signature**

```
( ) (success: boolean)
```

Defined in [Repository.ts:262](#)

Parameters

- **success:** *boolean*

update

```
( ) update(item: T in app.domain.repository.core.Repository<T extends PersistentEntity>, callback: Function)
```

Defined in [Repository.ts:289](#)

Update item in remote collection

Parameters

- **item:** *T in app.domain.repository.core.Repository<T extends PersistentEntity>*
 - The item to remove
- **callback:** *Function*
 - The callback is called with (true, item) on success and with (false, null) on error
- **callback function signature**

```
( ) (success: boolean, item: T in app.domain.repository.core.Repository<T extends PersistentEntity>)
```

Defined in [Repository.ts:289](#)

Parameters

- **success:** *boolean*
- **item:** *T in app.domain.repository.core.Repository<T extends PersistentEntity>*

- Container, dynamic module
- Enumeration
- Enumeration member
- Object literal
- Constructor
- Variable
- Function, call signature, accessor
- Index signature
- Interface
- Constructor
- Property
- Member, accessor
- Index signature
- Class
- Constructor
- Property
- Member, accessor
- Index signature
- Static property
- Static member

Generated using [TypeDoc](#)

The screenshot shows a web IDE interface with a sidebar on the left and a main workspace. The sidebar contains three sections: 'Index', 'Modules', and 'app'. The 'app' module is expanded, showing 'app: app' and 'Defined in TemplateProcessor.ts:1'. Below this, there is another 'Index' section, followed by 'Modules' and 'service'. The 'service' module is expanded, showing 'service: service' and 'Defined in TemplateProcessor.ts:1'. The main workspace is currently empty.

The screenshot shows a web IDE interface with a sidebar on the left and a main workspace. The sidebar contains three sections: 'Index', 'Classes', and 'ProcessorPattern'. The 'ProcessorPattern' class is expanded, showing 'ProcessorPattern: ProcessorPattern' and 'Defined in TemplateProcessor.ts:2'. Below this, there is a 'name' section, followed by 'name.pattern', and 'name.postSignLength'. The main workspace is currently empty.

Index

Classes

- ProcessorPattern
- TemplateProcessor
- VariablePattern

Classes

ProcessorPattern

ProcessorPattern: *ProcessorPattern*

Defined in [TemplateProcessor.ts:2](#)

name

- name: *Object*

Defined in [TemplateProcessor.ts:4](#)

name.pattern

- name.pattern: *string*

Defined in [TemplateProcessor.ts:5](#)

name.postSignLength

- name.postSignLength: *number*

Defined in [TemplateProcessor.ts:9](#)

⚙️ ☰

- `name.preSignLength`: *number*

Defined in [TemplateProcessor.ts:7](#)

parameter

- `parameter`: *Object*

Defined in [TemplateProcessor.ts:11](#)

parameter.pattern

- `parameter.pattern`: *string*

Defined in [TemplateProcessor.ts:12](#)

parameter.postSignLength

- `parameter.postSignLength`: *number*

Defined in [TemplateProcessor.ts:16](#)

parameter.preSignLength

- `parameter.preSignLength`: *number*

Defined in [TemplateProcessor.ts:14](#)

pattern

- `pattern`: *string*

Defined in [TemplateProcessor.ts:3](#)

⚙️ ☰

- 🔗 `TemplateProcessor`: [TemplateProcessor](#)

Defined in [TemplateProcessor.ts:29](#)

constructor

- 🔗 `constructor(data: Object, template: string, processors: Object): TemplateProcessor`

Defined in [TemplateProcessor.ts:101](#)

Parameters

- **data**: *Object*
 - A dictionary used by variables and processors for template rendering
- **template**: *string*
 - A text template containing markers and processors to replace
- **processors**: *Object*
 - A dictionary with processor functions
- **processors[]**

[] (index: *string*): *any*

Defined in [TemplateProcessor.ts:101](#)

Parameters

 - **index**: *string*

Returns

any

Returns [TemplateProcessor](#)

primaryProcessorPattern

- `primaryProcessorPattern`: [ProcessorPattern](#)

Defined in [TemplateProcessor.ts:67](#)

processor pattern: \$processorName:(param1, param2)\$

```
example : $processor:(abc1, abc, "ef")$ $processor:(path.to.variable,
"ab")$ $processor:(abc1)$ $processor:(abc1 )$
$processor:()$
```

primaryVariablePattern

- primaryVariablePattern: [VariablePattern](#)

Defined in [TemplateProcessor.ts:39](#)

```
example : ${path.to.something} ${variable}
```

secondaryProcessorPattern

- secondaryProcessorPattern: [ProcessorPattern](#)

Defined in [TemplateProcessor.ts:80](#)

secondaryVariablePattern

- secondaryVariablePattern: [VariablePattern](#)

Defined in [TemplateProcessor.ts:45](#)

parseProcessors

```
() parseProcessors(text: string, processorPattern:
ProcessorPattern, executer: Function): string
```

Defined in [TemplateProcessor.ts:154](#)

Parse processors and fetch text to replace the processor tags

Parameters

- text:** *string*
 - Template to search for processors

executer: Function

- Function to be execute for every found processor - should return the text which will replace the processor tag

executer function signature

```
() (processorName: string, processorParameters:
Array<string>, startIndex: number, length:
number): string
```

Defined in [TemplateProcessor.ts:155](#)

Parameters

- processorName:** *string*
- processorParameters:** *Array<string>*
- startIndex:** *number*
- length:** *number*

Returns *string*

Returns *string*

- Template with replaced processor tags

parseVariables

```
() parseVariables(variablePattern: VariablePattern, text:
string): string
```

Defined in [TemplateProcessor.ts:276](#)

find patterns like \${var} or \${var.auto.name} or \${!{var}}

Parameters

- variablePattern:** [VariablePattern](#)
- text:** *string*
 - The template containing variable markers to replace

Returns *string*

textToReplace - The template with replaced variable markers

`process(): string`

Defined in [TemplateProcessor.ts:112](#)

process member template using member data and member processors

Returns *string*

template - The rendered text template

processSecondary

`processSecondary(): string`

Defined in [TemplateProcessor.ts:131](#)

process member template using member data and member secondary processors

Returns *string*

template - The rendered text template

runProcessor

`runProcessor(processorName: string, processorParameters: Array<string>): string`

Defined in [TemplateProcessor.ts:190](#)

Execute a processor

```
example : runProcessor('concat', ["path.to.title", "\n:""",
"path.to.value"]);
```

Parameters

- **processorName:** *string*
- **processorParameters:** *Array<string>*
 - A list with string and path variables

string variables: start and end with `"`, e.g.

path variables: e.g.
`"path.to.variable"`

Returns *string*

- The rendered processor or `""`

VariablePattern

`VariablePattern: VariablePattern`

Defined in [TemplateProcessor.ts:19](#)

pattern

- **pattern:** *string*

Defined in [TemplateProcessor.ts:20](#)

postSignLength



- **postSignLength:** *number*




















Defined in [TemplateProcessor.ts:24](#)

preSignLength

- **preSignLength:** *number*

Defined in [TemplateProcessor.ts:22](#)

<ul style="list-style-type: none">  Container, dynamic module  Object literal  Constructor  Variable  Function, call signature, accessor  Index signature  Class  Constructor  Property  Member, accessor  Index signature 	<ul style="list-style-type: none">  Enumeration  Enumeration member  Interface  Constructor  Property  Member, accessor  Index signature
<ul style="list-style-type: none">  Static property  Static member 	

Generated using [TypeDoc](#)

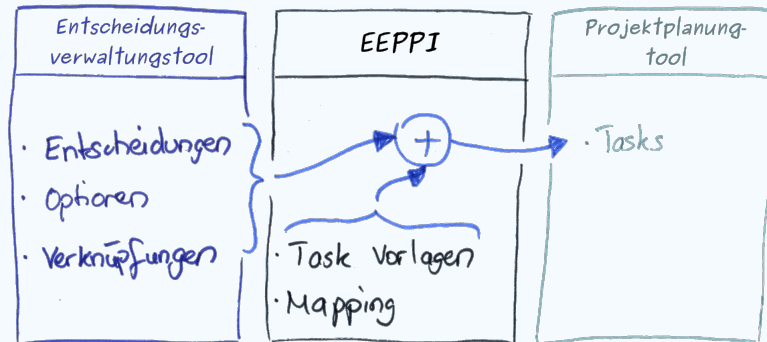


EEPPi Entwurfsentscheidungen als Projektplanungsinstrument

«Lassen sich aus Projektentscheidungen Aufgaben ableiten?»

Ausgangslage

- Projekte erfordern Entscheidungen
- Entscheidung resultieren häufig in ähnliche Aufgaben
- Für Entscheidungsverwaltung existieren Werkzeuge
- Für Projektplanung existieren Werkzeuge
- EEPPi will Entscheidungsmanagement und Projektplanung verbinden



EEPPi

- Webapplikation
- Bezieht Entscheidungen aus Wissensverwaltungssystem
- Metamapping zum Zuordnen von Aufgaben an Entscheidungen
- Überträgt Aufgaben an Projektplanungstool

- zu generierende Aufgaben frei konfigurierbar
- Templatingmechanismus für maximale Flexibilität
- Prozessoren für intelligente Templates

Fazit: EEPPi zeigt...

- ...Zukunft (was alles möglich ist)
- ...Design-Herausforderungen (hohe Flexibilität und Konfigurierbarkeit)
- ...Entwurfsentscheidungen lassen sich für Projektplanung nutzen