

**Hochschule Rhein-Waal**

Fakultät: Kommunikation und Umwelt

**Konzeption und Entwicklung eines  
Systems zur softwaregestützten  
Dokumentation von  
Unternehmensstrukturen für  
automatisierte Fortschrittsmessung und  
Werteorientierung**

Bachelorarbeit

vorgelegt von

Maximilian Oedinger

Hochschule Rhein-Waal  
Fakultät: Kommunikation und Umwelt

betreuer Professor:  
Herr Prof. Dr. Thomas Richter

---

**Konzeption und Entwicklung eines Systems zur  
softwaregestützten Dokumentation von  
Unternehmensstrukturen für automatisierte  
Fortschrittsmessung und Werteorientierung**

Bachelorarbeit  
im Studiengang  
Medieninformatik  
zur Erlangung des akademischen Grades

**Bachelor of Science**

---

vorgelegt von  
**Maximilian Oedinger**

En de Bongert 7

47918 Tönisvorst

Matrikelnummer:  
25208

Abgabedatum:  
(Due Date goes here)

## Zusammenfassung

Agility plays a big role in managing the complexity of volatile and unpredictable environments, such as project management and project portfolio management, by focussing on the value created for the customer. This leads to the need of scaling agile practices to enterprise level, which can be done by using i.e. the Scaled Agile Framework (SAFe) or Flight Levels.

For effective decision making in those environments, decisions should be made with metrics in mind that are relevant for the Level in which the decision must be made. This work presents an all in one customizable software solution to aggregate such metrics to the level required and present it in a meaningful manner, providing insights on progress from levels below the one where the decision must be made and what implications the decision will have on the levels above. Focus of this work is the adaptability of the solution regardless of the framework used to scale agile practices.

The solution was evaluated in a case study to determine its usefulness of the software. The results show that the adaptability of the solution is given, but to be useful in a real world scenario, the solution needs to be extended with time-based metrics like velocity, etc. and the ability to not only import data from operational tools like Jira, but also a synchronization of data between the tools, to make the solution usable over time, since the data only represents a snapshot of the current state of the, if the data is not updated regularly.

**Keywords:** agile portfolio management, agile project management, progress tracking, decision making, Flight Levels, Scaled Agile Framework, value based software engineering, data aggregation, visualization, web development, mevn stack

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>iv</b>
<b>Symbolverzeichnis</b>	<b>v</b>
<b>Abbildungsverzeichnis</b>	<b>vi</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	1
1.3 Methodik . . . . .	1
1.4 Gliederung der Arbeit . . . . .	2
<b>2 Agile Unternehmensführung</b>	<b>3</b>
2.1 Agilität . . . . .	3
2.2 Agile Projektstruktur . . . . .	4
2.3 Agiles Portfoliomanagement . . . . .	6
2.4 Agile Unternehmensstrukturierung . . . . .	6
2.4.1 SAFe . . . . .	6
2.4.2 Flight-Level . . . . .	7
<b>3 Analyse</b>	<b>9</b>
3.1 Reporting für agiles Portfoliomanagement . . . . .	9
3.2 qualitatives vs. quantitatives Reporting . . . . .	9
3.3 Reports für Value based Software-Engineering . . . . .	10
3.4 Automatisches Reporting . . . . .	11
3.5 Reporting mit digitalen Tools . . . . .	11
3.5.1 Verwendung von digitalen Tools . . . . .	11
3.5.2 Existierende Tools . . . . .	11
<b>4 Konzeption</b>	<b>12</b>
4.1 Prozessdefinition / Anforderungsformulierung . . . . .	12
4.2 UX-Entwurf für die Abbildung des Prozesses . . . . .	14
4.2.1 Authentifizierung . . . . .	15
4.2.2 Menü . . . . .	16
4.2.3 Organisation . . . . .	16

4.2.4	Dashboard . . . . .	17
4.2.5	Levels . . . . .	17
4.2.6	Gruppen . . . . .	18
4.2.7	Projekt . . . . .	18
4.3	Datenaggregation . . . . .	20
<b>5</b>	<b>Implementierung</b>	<b>22</b>
5.1	Datenfluss . . . . .	22
5.2	Datenstruktur . . . . .	23
5.3	Backend-Architektur . . . . .	24
5.4	Benutzeroberfläche . . . . .	29
5.4.1	Layouts . . . . .	29
5.4.2	Views und Components . . . . .	30
5.4.3	Datenverwaltung . . . . .	31
5.5	Projekt-Import . . . . .	34
5.6	Visualisierung/Datendarstellung . . . . .	37
5.7	lokale Inbetriebnahme . . . . .	40
5.8	CI/CD . . . . .	40
<b>6</b>	<b>Evaluation</b>	<b>44</b>
6.1	Praxistest . . . . .	44
6.2	Optimierungsvorschläge . . . . .	44
<b>7</b>	<b>Fazit</b>	<b>45</b>
7.1	Ergebnis . . . . .	45
7.2	Reflexion . . . . .	45
7.3	Ausblick . . . . .	45
	<b>Literaturverzeichnis</b>	<b>46</b>
<b>A</b>	<b>Anhang</b>	<b>49</b>
A.1	Anhang 1 . . . . .	49
	<b>Selbständigkeitserklärung</b>	<b>50</b>

## **Abkürzungsverzeichnis**

Abkürzung	Erklärung
-----------	-----------

## Symbolverzeichnis

Symbol	Erklärung

# Abbildungsverzeichnis

1	Agile Projektsegmentierung . . . . .	5
2	UX-Prototyp für die Anwendung . . . . .	15
3	UX-Prototyp für die Authentifizierung . . . . .	16
4	UX-Prototyp für die Organisationsseite . . . . .	17
5	UX-Prototyp für das Dashboard . . . . .	17
6	UX-Prototyp für die Levelübersicht . . . . .	18
7	UX-Prototyp für die Gruppenübersicht . . . . .	18
8	UX-Prototyp für das Aufgaben-Board . . . . .	19
9	UX-Prototyp für das Backlog . . . . .	20
10	UX-Prototyp für die Epics-Übersicht . . . . .	20
11	Prozess für Fortschrittsaggregation . . . . .	21
12	Datenfluss der Anwendung . . . . .	22
13	Datenfluss der Anwendung inkl. Jira . . . . .	23
14	UML-Diagramm der Datenstruktur . . . . .	23
15	Backend Architektur . . . . .	25
16	Abbildung der BE-Struktur . . . . .	26
17	API-Dokumentation . . . . .	28
18	Visualisierung des Layouts . . . . .	30
19	Import Feld Mapping . . . . .	34
20	Import Jira Projekt Auswahl . . . . .	35
21	Import Jira Ansicht nach Projekt Auswahl . . . . .	35
22	Import Jira Mapping . . . . .	36
23	Import Status Mapping . . . . .	36
24	Import Übersicht bei Excel Datei . . . . .	37
25	Import Übersicht bei Jira Import . . . . .	37
26	Dashboard . . . . .	38
27	Dashboard mit scrollbaren Gruppen . . . . .	39
28	Dashboard mit hervorgehobenen Gruppen . . . . .	39

# 1 Einleitung

## 1.1 Motivation

Damit Prozessplanungen mit hoher Volatilität in ihren Anforderungen effektiv umgesetzt werden können, haben sich bereits agile Methoden etabliert. Diese Methoden werden häufig auch auf höhere Unternehmensebenen skaliert. Damit diese Skalierung erfolgreich sein kann und auf allen Ebenen wertgetriebene Entscheidungen getroffen werden können, müssen Metriken für diese Ebenen ermittelt werden, welche eine Aggregation der darunterliegenden Planungselemente darstellen. Diese Aggregation über verschiedene Ebenen stellt eine Herausforderung dar, wenn in den verschiedenen Ebenen unterschiedliche Lösungen für die Dokumentation ihrer Arbeit und die Erfassung der Metriken verwendet werden, welche benötigt werden, um die Aggregation durchzuführen.

## 1.2 Zielsetzung

Das Ziel der Arbeit ist die Entwicklung eines Konzepts für die Dokumentation einer Unternehmensstruktur mit der Planungselemente aus verschiedenen Ebenen in einen Zusammenhang gebracht werden können, um über diese Zusammenhänge Metriken aggregieren zu können. Als Beispiel für diese Metriken wird der Fortschritt verwendet. Kern dieses Konzepts ist eine flexible Datenstruktur, welche das Ziel hat möglichst jede Unternehmensstruktur abzubilden, unabhängig von dem verwendeten Struktur-Framework und die Visualisierung dieser Datenstruktur. Dieses Konzept wird anschließend prototypisch implementiert und validiert.

## 1.3 Methodik

Durch eine Literaturanalyse Projekt und Projekt-Portfoliomanagement insbesondere im Kontext der Agilität erläutert. Dabei wurde ein Kontext zwischen der Skalierung von agilen Methoden auf höhere Unternehmensebenen geschaffen und dargestellt welche Rolle Metriken und das dafür benötigte Reporting für höhere Unternehmensebenen spielen. Anschließend wurde untersucht, wie wertebasierte Entscheidungsfindung in verschiedenen Unternehmensbereichen funktioniert und wie diese in einer gesamten Unternehmensstruktur Anwendung finden. Zuletzt wurde auf Basis dieser Erkenntnisse ein Konzept für einen webbasierten Prototyp entwickelt und

implementiert, welches die Dokumentation relevanter Informationen beliebig strukturierter Unternehmen erlaubt und somit automatisiertes Reporting am Beispiel der Fortschrittsmessung ermöglicht, welcher anschließend evaluiert wurde.

## 1.4 Gliederung der Arbeit

Kapitel 2 und 4 erläutern Projekt und Projekt-Portfoliomangement vor allem in Kontext der Agilität und wie Reporting und daraus resultierende Metriken in der Entscheidungsfindung in verschiedenen Unternehmensebenen verwendet werden.

Aus diesen Erkenntnissen beschreibt Kapitel 4 die Entwicklung des Konzepts. Hierzu wird ein Prozess erarbeitet, welcher die Dokumentation beschreibt und Anforderungen formuliert, welche durch das Konzept erfüllt werden sollen. Anschließend wird ein UX-Konzept beschrieben, welches die Nutzerinteraktion mit der Datenstruktur in Form einer Weboberfläche abbildet. Zuletzt wird ein Algorithmus für die Aggregation von Metriken am Beispiel des Fortschritts entwickelt.

Kapitel 5 dokumentiert die Implementierung des Konzepts in Form eines Prototyps. Hier wird zunächst die Datenstruktur und die Architektur sowohl des Backends für die Verwaltung dieser Datenstruktur, als auch die Architektur des Frontends für die Nutzerinteraktion und Visualisierung dieser Datenstruktur beschrieben. Zuletzt wird die Inbetriebnahme des Prototyps erläutert.

Für die Evaluation wird in Kapitel 6 der Praxistest beschrieben, welcher die Anwendung des Prototyps mit praxisnahen Daten simuliert und somit Feedback eines Experten ermöglicht.

Abschließend wird in Kapitel 7 eine kritische Reflexion auf das erarbeitete Konzept und den implementierten Prototypen gegeben und ein Ausblick auf mögliche Weiterentwicklungen auf Basis des Ergebnisses der Evaluation gegeben.

## 2 Agile Unternehmensführung

Unternehmen haben strategische Ziele welche erreicht werden müssen, um Konkurrenzfähigkeit zu erhalten. Um Ziele zu erreichen werden Prozesse wie z. B. Projekte umgesetzt. Um diese zu steuern und zu kontrollieren, ist eine Form von Management notwendig, die passend zu der Unternehmensgröße und der Arbeitsweise des Unternehmens gewählt ist. Im Folgenden werden verschiedenen Anforderungen an das Management von Unternehmen dargestellt und die daraus resultierenden Methoden und Konzepte erläutert.

### 2.1 Agilität

Agilität im Kontext von Projektmanagement oder auch grundsätzlicher Unternehmensorganisation ist ein alternativer Ansatz für die Planung unternehmensinterner Prozesse, wie z. B. die Umsetzung eines Projekts und steht meist dem sogenannten traditionellen Ansatz gegenüber. Unter diesem traditionellen Ansatz wird für gewöhnlich der lineare Planungsprozess verstanden, welcher voraussetzt, dass alle Anforderungen vor der Umsetzung klar und eindeutig definiert und dokumentiert sind und somit Risiko minimiert wird. Diese Umstände sind allerdings häufig nicht gegeben, bevor ein geplanter Prozess beginnen muss, damit Konkurrenzfähigkeit für ein Unternehmen erhalten werden kann. Solche zeitkritischen Prozesse sind häufig aber maßgebend für den Erfolg eines Unternehmens, wodurch ein Bedarf für eine Methode entstand, die Anpassbarkeit an sich ändernde Anforderungen und Rahmenbedingungen während der Umsetzung eines Prozesses erlaubt [1].

Bei traditioneller Planung erhöht sich durch sich ändernde Anforderungen das Risiko die falschen Dinge zum falschen Zeitpunkt zu tun. Agile Methodik dagegen erlaubt es diese Prozesse so effektiv wie möglich zu managen, da die Planung nicht linear, sondern iterativ stattfindet, sodass dieses Risiko immer nur für eine Iteration in Kauf genommen werden muss. Durch regelmäßige Feedbackschleifen mit Stakeholdern bleibt der Fokus auf Werteorientierung, da sich ändernde Anforderungen regelmäßig bereits in den Planungsprozess der nächsten Iteration einbezogen werden. Somit entsteht eine Flexibilität und Anpassbarkeit, welche die Volatilität verringert. In agilen Projektteams gibt es keinen Projektmanager. Dies bedeutet aber nicht das es nicht die Aufgaben gibt, die typischerweise von einer solchen Rolle übernommen würden. Stattdessen werden Planungs- und Entscheidungsprozesse in-

nerhalb des Teams von dem Team selbst übernommen. Dies setzt hoch qualifizierte und selbstorganisierte Teams voraus. Somit werden die Aufgaben eines traditionellen Projektmanagers in das Team übergeben und unter den Mitgliedern aufgeteilt oder von allen gemeinsam übernommen [2]. Dadurch, dass Dinge erst dann entschieden werden, wenn es notwendig ist, ist allerdings der Gesamtaufwand und die -dauer nicht zu Beginn einschätzbar, sondern immer nur der Aufwand und die Dauer der aktuellen Iteration [1].

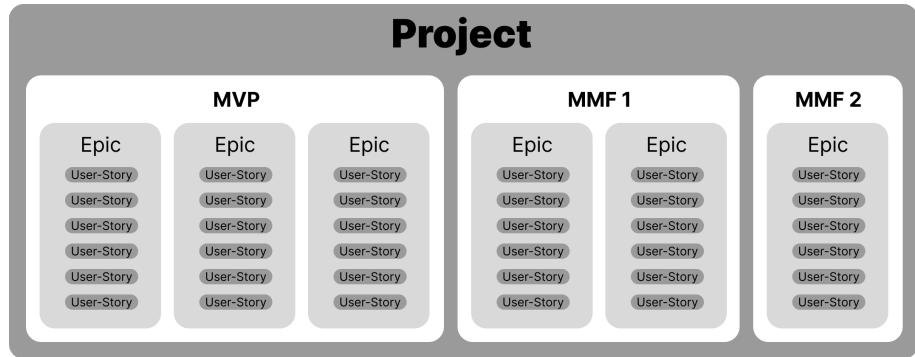
Ziel bei der Wahl der Planungsmethode ist immer den Erfolg der Umsetzung des geplanten Prozesses zu maximieren. Die richtige Wahl der Methode sollte zu Eingrenzung des Projektumfangs, schneller Lieferung, Qualitätssicherung, Kundenzufriedenheit und klarer Kommunikation an/zwischen Stakeholdern und somit zu Projekterfolg führen [3]. Dieser Projekterfolg kann in zwei Schlüsselfaktoren unterteilt werden: Projekteffizienz, welche die Kostenminimierung und die Erreichung von Projekt basierenden Zielen beschreibt, und Projekteffektivität, welche den Einfluss des Projekts auf die Erreichung größerer Unternehmensziele, die von Stakeholdern definiert werden, beschreibt [4]. Für Projekteffizienz, oder auch langfristiger Projekterfolg genannt, wurde bereits untersucht, inwiefern die Verwendung von agilen Methoden, den langfristigen Projekt erfolgt steigert. Dabei stellte sich heraus, dass gerade der Erfolg agiler Projektplanung von der Qualität der Teamarbeit abhängig ist. Außerdem zeigte sich, dass in den meisten Fällen ein hybrider Ansatz sowohl dem traditionellen als auch dem strikt agilen Ansatz überlegen ist [5]. *Hat ein Projekt keinen Bedarf für agile Vorgehensweisen und wird dennoch agil durchgeführt, kann dies zu Verminderung der Projekteffizienz führen.*

## 2.2 Agile Projektstruktur

Um die Ziele eines agilen Ansatzes im Projektmanagement zu erreichen, muss die Struktur auch dem agilen Umsetzungsprozess gewachsen sein. Dazu muss das Projekt mit einem iterativen Gedanken geplant werden. Das Projekt muss also nicht nur als ein fertiges Produkt welches am Ende der vollständigen Umsetzung entstanden ist, sondern auch die Zwischenprodukte die am Ende jeder Iteration bereits entstanden sind, betrachtet werden. Das Ziel von agilem PM ist funktionierende Software und das so früh wie möglich, ohne das das vollständige Projekt umgesetzt werden muss. Hierzu verwendet man das Minimum Viable Product(MVP) und Minimum

Marketable Features (MMF) [6]. Mit diesen funktionierenden und in sich geschlossenen Iterationen kann während der Umsetzung regelmäßig ein Feedbackprozess mit einem funktionierenden Produkt stattfinden. Diese Feedbackschleifen sind essenziell für die Erhaltung von Werteorientierung und Anpassbarkeit im Verlauf des Projekts [6].

Für MVP und MMF muss das Projekt bis zu einer bestimmten Granularität segmentiert werden. Hierzu verwendet man typischerweise sogenannte Epics und User-Stories. Epics sind in sich geschlossene Bestandteile des Produkts. Mehrere dieser Epics bilden zusammen mit ihrer vollständigen Umsetzung ein MVP oder MMF. Epics können anschließend in User-Stories aufgespalten werden, welche konkrete Funktionalitäten beschreiben. [7]



**Abbildung 1:** Agile Projektsegmentierung

User-Stories können zum Zeitpunkt der Umsetzung erneut in sogenannte Tasks unterteilt werden. Tasks sind Arbeitspakete die effektiv von einer einzelnen Person an einem Stück bearbeitet werden können. Diese Arbeitspakete sollten von den operativen Projektteammitgliedern definiert werden, da diese die am besten abschätzen können, wie die Beschaffenheit der Tasks sein muss, um Teamressourcen, wie Kapazität und Kompetenzen innerhalb des Teams, optimal zu nutzen. [7]

Für die Umsetzung gibt es verschiedene Kommunikations-Frameworks. Die beiden meist vertretenen Frameworks sind Scrum und Kanban. Beide Frameworks bieten Möglichkeiten den iterativen Umsatzprozess mit regelmäßigen Meetings, die ein konkretes Ziel verfolgen, zu strukturieren [6].

## 2.3 Agiles Portfoliomangement

Projekt Portfoliomangement(PPM) wird traditionell dazu verwendet eine Portfolio bestehend aus Projekten innerhalb eines Unternehmens zu verwalten. Die Projekte eines Portfolios teilen sich Ressourcen und müssen koordiniert, sowie priorisiert werden [8]. Das Ziel dabei ist immer den finanziellen Wert über die Projekte eines oder sogar mehrerer Portfolios hinweg zu maximieren und sie dabei mit Unternehmenszielen zu verknüpfen, um die Strategie des Unternehmens in Hinsicht auf die innerhalb des Unternehmens verfügbaren Ressourcen zu verfolgen [9].

*multi agile project product and stuff* Die Verwaltung von agilen Projekten führte dazu, dass Agilität bis auf das PPM skaliert werden kann, um Entscheidungen auf Portfolioebene ebenfalls Wertegetrieben, statt ausschließlich Ressourcen- und damit Kosten-basiert, treffen zu können. Um dies zu erreichen, gibt es ebenfalls verschiedene Frameworks, die in den folgenden Abschnitten genauer beschrieben werden [10].

Vorteile der Implementierung von agilen Framework für PPM stellt bei Portfolios mit variablen und innovativen Projekten eine Möglichkeit dar, Time-to-Market, Team-Produktivität und Kommunikation zu steigern [11].

Eine Studie zeigte ebenfalls, dass sich durch diese Implementierung (1) Transparenz über Ressourcen und Aufgaben erhöht und damit Vertrauen, Entscheidungsfindung und Ressourcenzuweisung verbessert wird, (2) bessere Zusammenarbeit und routinierte Interaktion zu Artefakten für die Ermöglichung regelmäßiger Feedbackschleifen führt, (3) Commitment für die Strategie des Portfolios gesteigert wird und (4) die Orientierung des Teams fokussiert wird, indem Unruhen in der Ressourcenzuweisung reduziert und potenzial der Teams erhöht [12].

## 2.4 Agile Unternehmensstrukturierung

Für die Skalierung von Agilität auf die gesamte Unternehmensstruktur, gibt es, wie bereits erwähnt, verschiedene Frameworks. Diese werden im folgenden Abschnitt genauer beschrieben.

### 2.4.1 SAFe

Es wurde von Dean Leffingwell entwickelt und ist eine der meistverbreiteten Methoden für agile Unternehmensstrukturierung. SAFe unterteilt ein Unternehmen in drei

bis 4 Ebenen. Diese Ebenen sind:

- Team Level
- Program Level
- Value Stream Level (optional)
- Portfolio-Level

Das Team-Level bildet die operative Ebene in der agile Teams agieren. Diese Teams können SCRUM oder Kanban verwenden, besonders wert wird aber darauf gelegt, dass alle Teams in einem einheitlichen Iterationsintervall arbeiten.

Das Program Level fasst 5 bis 12 agile Teams aus dem Team-Level in ein virtuelles Programm zusammen, welches als Agile Release Train (ART) bezeichnet wird. Diese ARTs sind langlebige, selbstorganisierte Teams, welche gemeinsam für den Erfolg des ARTs verantwortlich sind.

Das optionale Value Stream Level ist eine Ebene, die verwendet werden kann, wenn es mehrere ARTs gibt, welche einen gemeinsamen Wertestrom ergeben und deshalb zusätzliche Koordination benötigen.

Das Portfolio-Level ist die Ebene, in der die Werteströme entweder aus dem Value Stream Level oder dem Program Level organisiert und finanziert werden. Dort finden Praktiken wie Lean-Agile Budgeting ihre Anwendung für die notwendige Kontrolle über die verschiedenen Werteströme.

Zuletzt gibt es noch die sog. Foundation Layer, welche verschiedene Richtlinien definiert, welche als Grundlage der Prinzipien hinter SAFe dienen. Dazu gehören die 9 Lean-Agile Principles, Leitlinien für agile Führungsrollen, etc. [13]

#### **2.4.2 Flight-Level**

Flight-Level ist eine weitere Methode, welche eine mögliche Form von agiler Unternehmensstrukturierung bietet und wurde von Klaus Leopold entwickelt. Die Methode Flight-Level wird hierbei von Leopold [14] als Kommunikations-Framework bezeichnet und unterteilt ein Unternehmen in 3 Ebenen oder auch Flight-Level, also Flughöhen bezeichnet:

Flight-Level 1: Operative Ebene mit einem Projekt oder Team

- Flight-Level 2: Koordinierung der Zusammenarbeit mehrerer Teams
- Flight-Level 3: Strategisches Portfoliomanagement.

Die verschiedenen Flight-Level sollen einen Blick in unterschiedlichen Detailgraden auf das Unternehmen ermöglichen, ähnlich wie bei einem Blick aus einem Flugzeug in verschiedenen Flughöhen. Kommunikation zwischen den Ebenen wird als kritischer Faktor dargestellt, welcher die Wertschöpfung steigern, da sich das Unternehmen besser und schneller an die Veränderungen des Marktes anpassen können soll und somit die Konkurrenzfähigkeit bewahren kann. Ähnlich wie bei agilen Projektmanagement-Frameworks, gibt diese Methode eine klare Struktur in der Kommunikation vor.

## 3 Analyse

Im vorherigen Kapitel wurde bereits untersucht wie Projekt und Projekt-Portfolio-management insbesondere in Kombination mit agiler Methodik die Erreichung strategischer Unternehmensziele unterstützen kann. Durch die regelmäßig anstehenden Entscheidungen besteht der Bedarf für regelmäßiges und vollständiges Reporting, welches für die Entscheidungen herangezogen werden kann, um diese zu objektivieren.

### 3.1 Reporting für agiles Portfoliomangement

Studien zeigten bereits eine positive Korrelation zwischen erfolgreichem Portfoliomangement und sogenannter Project Portfolio Control (PPC). PPC wird durch drei Faktoren charakterisiert: Projektauswahl, Reporting und Stil der Entscheidungsfindungs [15]. Für die optimale Projektauswahl in PPC wurde bereits untersucht, dass die Entscheidungsfindung optimiert werden kann, indem die Metriken aus dem Reporting, welche für die Entscheidungsfindunge herangezogen werden, mithilfe eines fuzzy Analytic Hierarchy Process (AHP) für eine Priorisierung einzelner Elemente des Portfolios gewichtet werden und anschließen mit der fuzzy TOPSIS Methode in eine Reihenfolge gebracht werden [16]. Hieraus lässt sich ableiten, dass es keine generalisierbaren Metriken gibt, die für alle Unternehmen und Projekte gelten, sondern dass die Metriken für jedes Unternehmen und jedes Projekt individuell bestimmt werden müssen. Im Optimalfall sollten also grundsätzlich alle bzw. möglichst viele Metriken erhoben werden, um sie anschließend zu gewichten.

### 3.2 qualitatives vs. quantitatives Reporting

Für effektives Reporting müssen verschiedene Metriken erhoben werden, hierbei unterscheidet man allgemein in qualitatives und quantitatives Reporting. Qualitatives Reporting zeigt Chancen auf und bietet Kontext, während quantitatives Reporting das Quantifizieren von Elementen und Fortschritt sowie die Validierung von Zielen und geschaffenem Wert ermöglicht [].

### 3.3 Reports für Value based Software-Engineering

Value-based Software-Engineering(VBSE) ist eine Sammlung von Frameworks für die Entscheidungsfindung in der Softwareentwicklung. VBSE basiert auf der Annahme, dass die Entscheidungen in der Softwareentwicklung auf Basis von einem Kriterium, welches als Wert bezeichnet wird, getroffen werden sollten [1]. Wert kann hier auf verschiedene Arten definiert werden und kann in mehrere Teile heruntergebrochen werden. Beispiele für Wert sind:

- Nutzen
- Kosten
- Risiko
- Wert für den Kunden
- Wert für das Unternehmen.

Unter der Berücksichtigung des Werts können Entscheidungen wertezentriert getroffen werden. Diese Entscheidungen verteilen sich über den gesamten Software-Engineering-Prozess, welcher auch als VBSE Agenda bezeichnet wird [1]. Der Prozess kann folgende Teile beinhalten:

- Requirements Engineering
- Architecting
- Design und Entwicklung
- Verifizierung und Validation
- Planung und Kontrolle
- Risikomanagement
- Qualitätsmanagement
- Mitarbeitermanagement

### 3.4 Automatisches Reporting

Reporting ist meist ein manueller Prozess, welcher mit immer wiederkehrendem Aufwand verbunden ist, da die Metriken regelmäßig erhoben werden müssen. Um diesen Prozess des Reportings zu optimieren, sollten qualitative und quantitative Reports unterschiedlich betrachtet werden. Quantitative Metriken sind quantifizierbar, sodass der Prozess der Erhebung dieser Metriken bei vollständiger Dokumentation aller relevanter Daten automatisierbar ist. Werden diese Metriken dann automatisch erhoben sorgt dies für konsistente, regelmäßige, valide und aktuelle Ergebnissen [1].

Qualitative Metriken dagegen sind schwer automatisierbar, da sie häufig nicht auf objektiv erfassbaren Daten beruht. Zur Optimierung kann eine systematische Herangehensweise für die Bestimmung der Metriken definiert werden, um mit deren Hilfe mehr Konsistenz und Regelmäßigkeit zu gewährleisten. Des Weiteren kann man davon ausgehen, dass künstliche Intelligenz in Zukunft eingesetzt werden kann, um auch qualitative Metriken weitestgehend zu automatisieren [2].

## 3.5 Reporting mit digitalen Tools

### 3.5.1 Verwendung von digitalen Tools

[17]

### 3.5.2 Existierende Tools

## 4 Konzeption

In den vorherigen Kapiteln wurde beleuchtet, dass Agilität skaliert auf Unternehmensebene verschiedene Vorteile und Risiken mit sich bringt und auf regelmäßige ad hoc Entscheidungen in verschiedenen Ebenen des Unternehmens angewiesen ist, um effektive Ergebnisse zu erzielen. Diese Entscheidungen sollten auf Basis von Metriken aus generalisierten Reporting-Prozessen getroffen werden, wobei diese Reporting-Prozesse, wenn möglich, automatisiert werden sollten, um Konsistenz, Regelmäßigkeit, Validität und Aktualität zu gewährleisten. Tool basierendes Reporting hat sich in vielen Unternehmen als eine erfolgreiche Automatisierungsmethode bewiesen. Resultierend aus diesen Erkenntnissen wird an dieser Stelle ein Prozess definiert der ein Tool für einen Dokumentations- und Reporting-Prozess abbildet, aus denen anschließend ein UX-Konzept und im nächsten Kapitel eine prototypische Implementierung entwickelt werden kann.

### 4.1 Prozessdefinition / Anforderungsformulierung

Für Fortschrittsmessung und Werteorientierung müssen Zusammenhänge innerhalb eines Portfolios mit operativen Elementen, deren absoluter Fortschritt tatsächlich gemessen werden kann, dokumentiert werden können. Diese Verknüpfung muss über beliebig viele Ebenen stattfinden können, um möglichst universell und unabhängig von der Unternehmensstruktur und dem verwendeten Framework benutzbar zu sein. Operative Elemente, hier genannt Aufgabe, haben einen veränderbaren Status an dem festgestellt werden kann, ob sie fertig sind und somit für den gemessenen Fortschritt einbezogen werden müssen. Aufgaben besitzen zudem zwei numerische Werte: Storypoints und Value. Storypoints sollen als relativer Wert die Komplexität der Umsetzung, die mit einer Aufgabe verbunden ist darstellen, während Value den Mehrwert widerspiegelt, welcher durch die Erledigung der Aufgabe entsteht. Die Bezeichnung ist in diesem Fall mit Absicht unspezifisch gewählt, um weitere Komplexität zu vermeiden und den Scope dieser Arbeit zu verkleinern, da zwischen verschiedenen Formen von Mehrwert unterschieden werden kann, wie z. B. Business-, Customer-Value oder auch interner Mehrwert. Ziel dieser Werte ist, die Aufgaben vergleichbarer zu machen, da einige Aufgaben mehr Einfluss auf den Fortschritt haben können als andere. Aufgaben im Kontext des Prototyps würden somit die Funktion der User-Stories erfüllen.

Die zuvor bereits beschriebene weitere Unterteilung in Tasks spielt für die Fortschrittsmessung oder Wertorientierung und dementsprechend auch für ein Refinement keine Rolle, da Mehrwert und Fortschritt nur entsteht, wenn eine User-Story vollständig umgesetzt wurde und wird deshalb vernachlässigt.

Damit die spezifische Struktur eines Unternehmens abgebildet werden kann, muss es möglich sein, verschiedene Ebenen anzulegen, welche die hierarchische Struktur darstellen kann, welche tatsächlich in den Unternehmen verwendet wird. Innerhalb dieser Ebenen können Elemente angelegt werden, welche mit anderen Elementen in darüber liegenden Ebenen verknüpft werden können. Die unterste Ebene ist immer die operative Ebene, welche für gewöhnlich als Projekt bezeichnet wird. Optional können Epics verwendet werden, um Aufgaben innerhalb eines Projekts zu gruppieren, sodass mehrere Aufgaben zusammengefasst z. B. ein Feature o. ä. darstellen können.

Der Fortschritt eines Projekts resultiert aus dem Verhältnis von offenen zu erledigten Aufgaben oder Epics, wobei Storypoints und Mehrwert als Faktoren hinzugezogen werden können, um das Verhältnis zu relativieren.

Der Fortschritt eines Planungselements, welches keine Aufgabe oder Projekt ist, wird durch den Gesamtfortschritt der verknüpften Elemente aus der darunterliegenden Ebene aggregiert. Daraus ergibt sich eine Baumartige Struktur, welche die gesamte Planungsstruktur eines Unternehmens abbilden soll.

Um die Evaluierung zu erleichtern und mehrere Szenarien testbar zu machen, können mehrere dieser Strukturen innerhalb der Anwendung existieren, weshalb es eine Ebene gibt, die hier Organisation genannt wird. Organisationen können unabhängig voneinander existieren und eine vollständige Datenstruktur beinhalten.

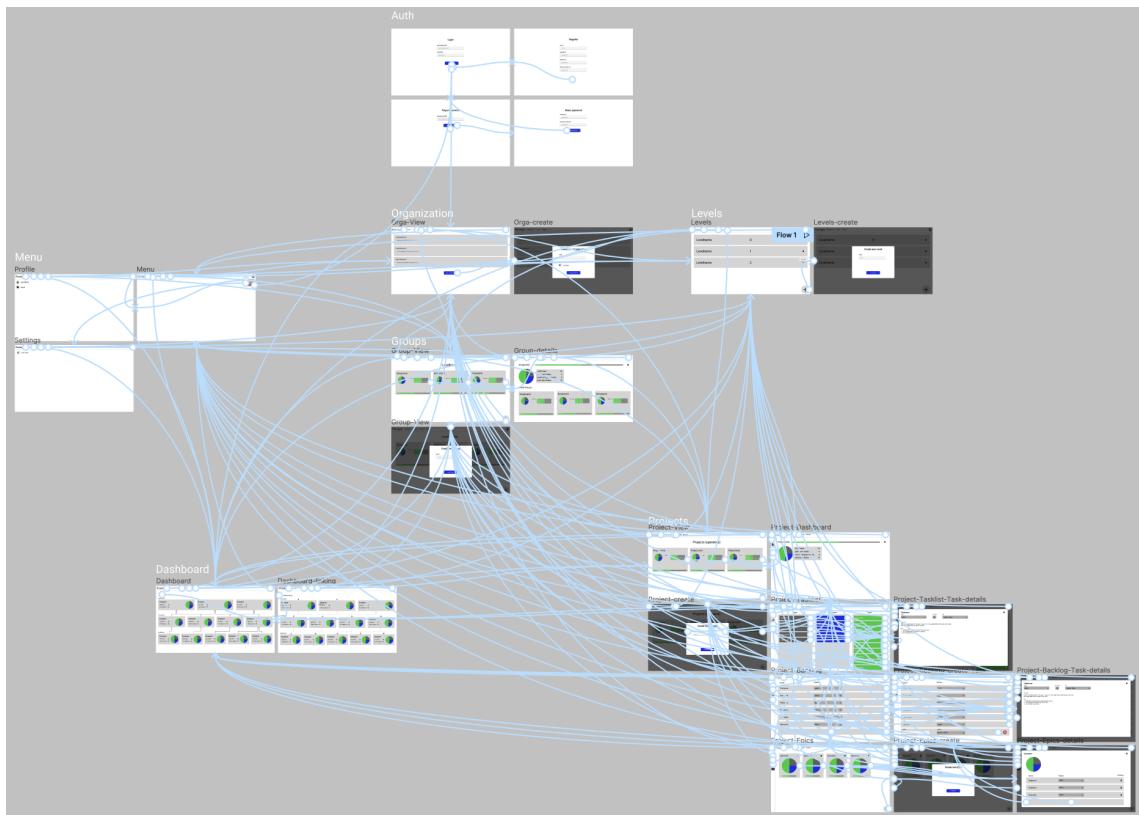
Um die Adaption des Tools zu erleichtern, sollte es eine Schnittstelle geben über die der Nutzer Daten aus anderen Tools, insbesondere aus der operativen Ebene, importieren kann. Diese Schnittstelle sollte es also erlauben Projekte bzw. vor allem die Aufgaben aus Projekten, inklusive ihres Status, Epics, Storypoints und Value zu importieren. Diese Schnittstelle wird aus Gründen der Komplexität und des Scopes dieser Arbeit nicht als Synchronisation implementiert, sondern als einmaliger Import. Sodass zumindest ein aktueller Stand der Daten importiert werden kann, um die Anwendung einmalig zu testen. Hier ist bereits klar, dass das Tool nur durch die Synchronisation mit anderen Tools sinnvoll produktiv eingesetzt werden kann, da das Ziel dieser Arbeit nicht die Entwicklung eines besseren Projektmanagement-

Tools ist. Die tatsächliche Verwaltung der Aufgaben wird besser in andere Tools abgebildet, die Inklusion eines Task-Boards und eines Backlogs sind an dieser Stelle nur notwendig, damit der Prototyp auch ohne andere Tools effektiv verwendet werden kann.

Das Ziel des Prototyps einen zentralen Ort für alle relevanten Informationen für die Entscheidungsfindung auf verschiedenen Ebenen zu schaffen. Dabei darf allerdings nicht außer Acht gelassen werden, dass es Informationen in einem Unternehmen geben kann, die nicht transparent für alle Mitarbeiter zugänglich sein sollten. Deshalb wird eine Authentifizierung und vor allem eine Autorisierung benötigt, mit der festgestellt werden kann, ob ein Nutzer diese Information sehen darf, bzw. in welchem Umfang die Informationen zur Verfügung gestellt werden dürfen. Um den Scope dieser Arbeit einzuschränken, soll eine nur Authentifizierung als Grundlage implementiert werden, allerdings ist für die Autorisierung eine Nutzerverwaltung notwendig, welche nicht implementiert wird, aber hinzugefügt werden kann.

## 4.2 UX-Entwurf für die Abbildung des Prozesses

Um den Userflow innerhalb des Prototyps darzustellen, wurde ein UX-Entwurf mit Figma entwickelt, welcher alle Funktionalitäten der Anwendung visuell abbildet und als Grundlage für das Design der Benutzeroberfläche dient. Der Entwurf ist als Klickprototyp hier verfügbar.

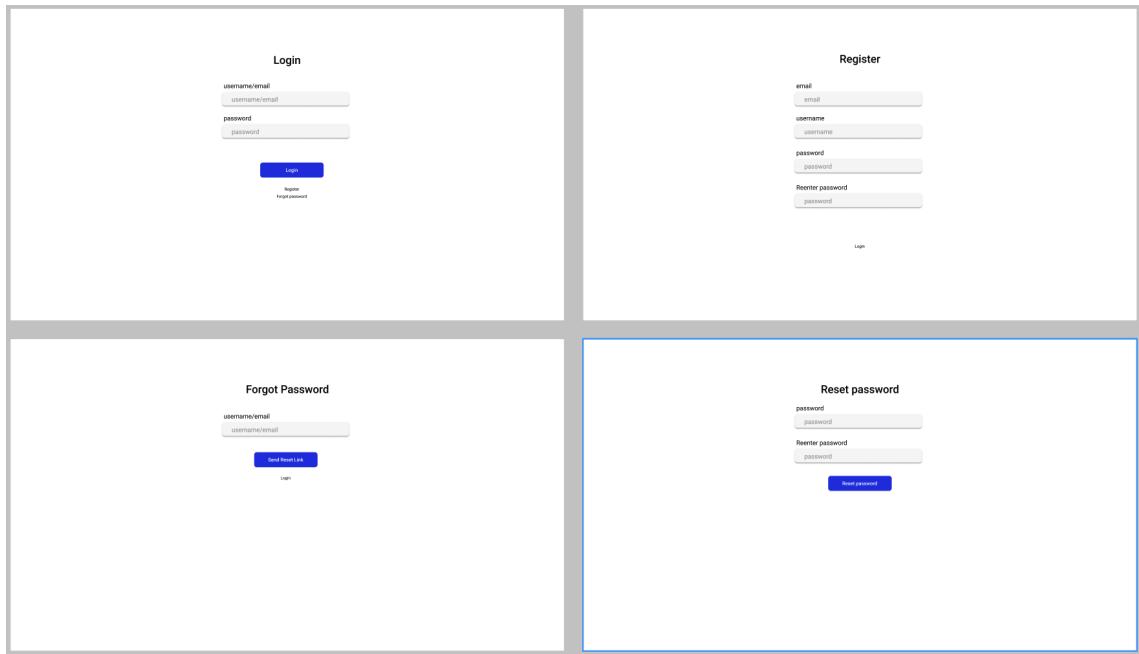


**Abbildung 2:** UX-Prototyp für die Anwendung

Der Entwurf unterteilt die Anwendung in 7 verschiedene Teile:

#### 4.2.1 Authentifizierung

Die Authentifizierung beinhaltet Seiten für den Login und die Registrierung. Außerdem muss der Nutzer sein Passwort zurücksetzen können, indem er einen Reset-Link für seine registrierte E-Mail-Adresse anfordert und über diesen Link ein neues Passwort vergeben kann.



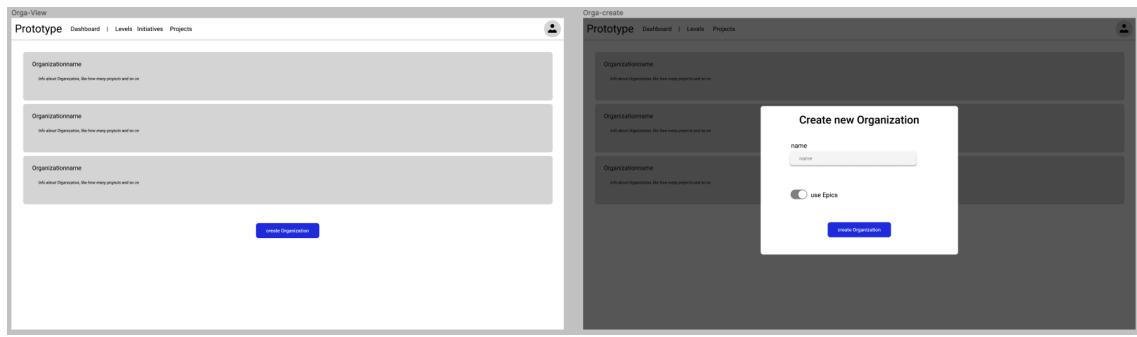
**Abbildung 3:** UX-Prototyp für die Authentifizierung

#### 4.2.2 Menü

Das Menü besteht aus 2 Seiten. Die Profil-Seite stellt den Nutzernamen und E-Mail-Adresse dar und ermöglicht es dem Nutzer ein JIRA-API-Token für einen JIRA-Import zu hinterlegen. Auf der Einstellungsseite kann der Nutzer zwischen Light- und Dark-Mode wechseln.

#### 4.2.3 Organisation

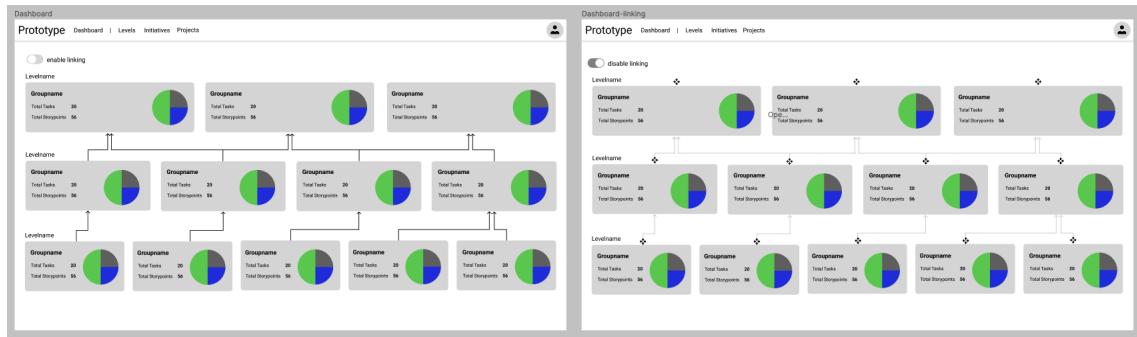
Auf der Organisationsseite kann der Nutzer alle Organisationen sehen und neue Organisationen erstellen. Beim Erstellen einer Organisation kann der Nutzer sich entscheiden, ob er Epics verwenden möchte. Wenn die Organisation erstellt wird, wird zusätzlich ein Level für Projekte, also die unterste Ebene erstellt. Verwendet die erstellte Organisation Epics, werden 2 Ebenen standardmäßig erstellt: Project und Epic. Dies hat ebenfalls Auswirkungen auf das Dashboard, da es grundsätzlich alle Levels darstellt. Verwendet eine Organisation allerdings Epics, wird die unterste Ebene, also Epics, nicht mit dargestellt. Außerdem gibt es in den Projekten keine Möglichkeit Epics zu erstellen.



**Abbildung 4:** UX-Prototyp für die Organisationsseite

#### 4.2.4 Dashboard

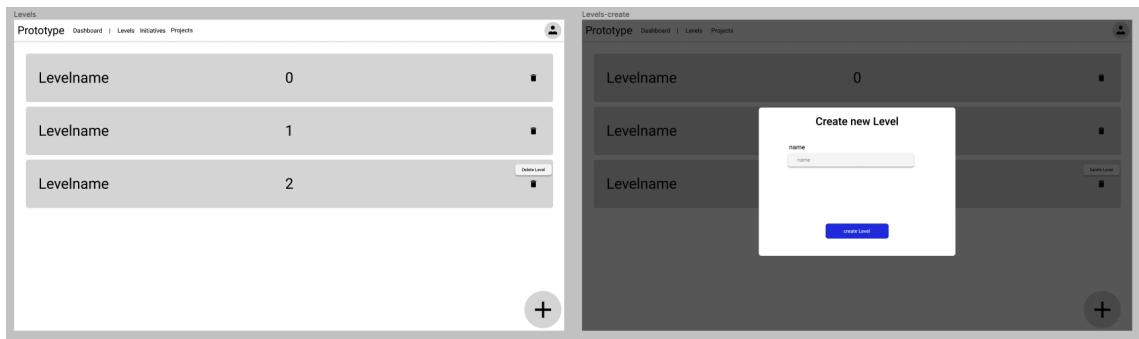
Das Dashboard zeigt alle Planungselemente einer Organisation hierarchisch angeordnet dar und wie diese miteinander verknüpft sind. Der Nutzer kann je Element anhand einer kleinen Grafik den aktuellen Fortschritt des Elements ablesen. Außerdem hat der User die Möglichkeit den Verlinkungsmodus auszuwählen, wodurch er mit Drag-and-drop neue Verknüpfungen erstellen kann. Durch einen Klick auf das Element gelangt der Nutzer zur Detailansicht des Elements.



**Abbildung 5:** UX-Prototyp für das Dashboard

#### 4.2.5 Levels

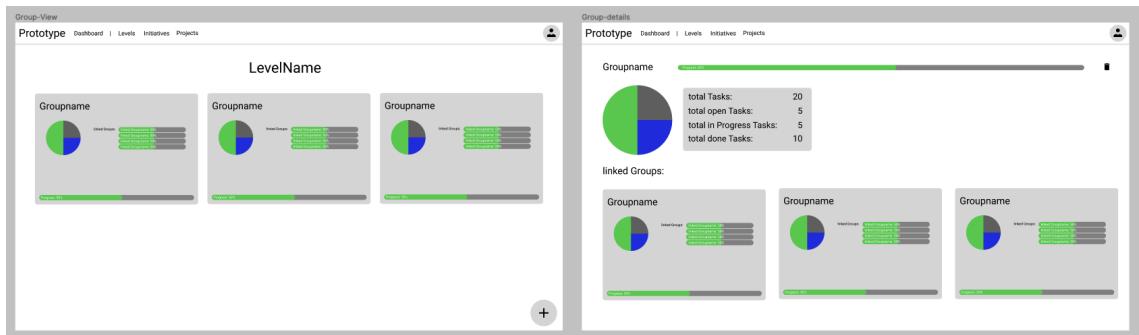
Hier kann der Nutzer Ebenen(Levels) erstellen und löschen. Alle Ebenen werden hierarchisch sortiert dargestellt.



**Abbildung 6:** UX-Prototyp für die Levelübersicht

#### 4.2.6 Gruppen

Je Ebene gibt es eine Gruppenansicht, die alle Elemente (Gruppen) je Level in mit einer kurzen Übersicht über den Fortschritt enthält. Der Nutzer kann neue Gruppen erstellen und auf bestehende Gruppen klicken um zu deren Detailansicht zu gelangen. In der Gruppendetailansicht gibt es eine detaillierte Übersicht über den Fortschritt des Elements. Der Nutzer kann das Element umbenennen und löschen. Außerdem werden alle verknüpften Elemente aus der darunterliegenden Ebene dargestellt. Mit einem Klick auf eines dieser Elemente kann der Nutzer auch deren Detailansicht aufrufen.



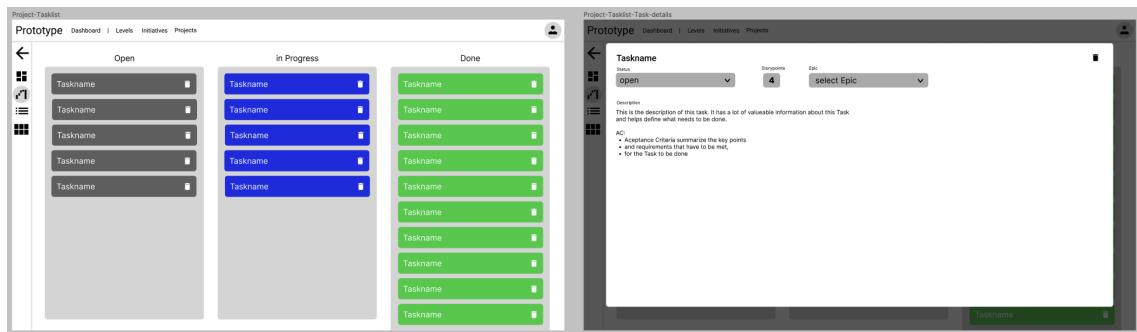
**Abbildung 7:** UX-Prototyp für die Gruppenübersicht

#### 4.2.7 Projekt

Die Projektübersicht ähnelt der Gruppenansicht, bietet aber zusätzlich zur einfachen Erstellung neuer Elemente auch die Möglichkeit eines Imports mit einer Excel-Datei. Außerdem kann ein Nutzer, der ein gültiges JIRA-API-Token in seinem Profil gespeichert hat, JIRA-Projekte importieren. Bei einem Import, unabhängig, ob aus

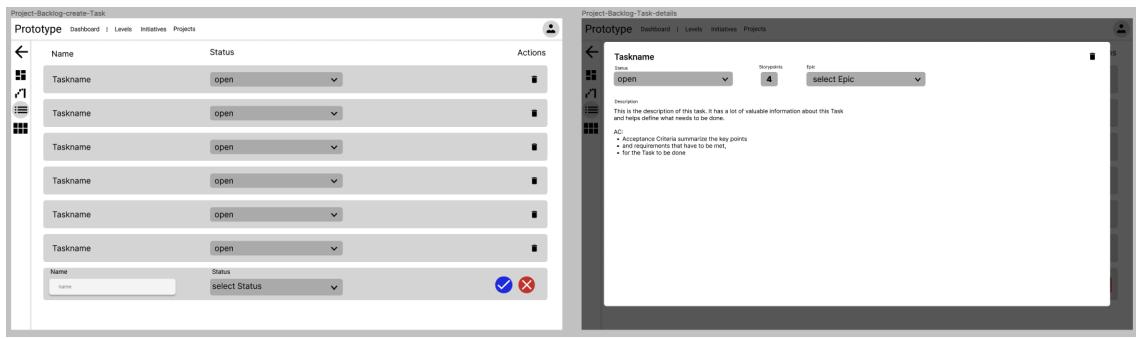
einer Excel-Datei oder von JIRA, wird nicht nur ein Projekt-Element erstellt, sondern ebenfalls Aufgaben innerhalb des Projekts, die aus der Excel-Datei oder JIRA ausgelesen wurden (Wie dieser Import aussieht und funktioniert war zum Zeitpunkt der Erstellung des UX-Prototyps noch nicht klar. Genauere Details hierzu werden später in der Implementierung erläutert). Durch einen Klick auf ein Element innerhalb der Übersicht gelangt der Nutzer ebenfalls in eine Detailansicht, welche sich allerdings stark von der einer gewöhnlichen Gruppe unterscheidet.

Der Nutzer kann ein Aufgaben-Board öffnen, in dem er alle Aufgaben des Projekts in drei Spalten sehen kann. Jede Spalte stellt einen der drei Fortschritts-Status dar: *open*, *in progress* und *done*. Der Nutzer kann mit Drag-and-drop den Status der Aufgaben ändern, indem er sie in die entsprechende Spalte bewegt. Jedes Element besitzt ein Löschesymbol, mit dem das Element, nach einer Bestätigung, gelöscht werden kann. Durch einen Klick auf ein Element öffnet sich ein Pop-up-Fenster mit der Detailansicht der Aufgabe. Diese stellt den Aufgabennamen, die Beschreibung, Storypoints und Value sowie ggf. das zugeordnete Epic dar. Möchte der Nutzer das Epic der Aufgabe wechseln oder sie einem Epic zuordnen, kann er mit einem Drop-down-Menü aus einem der erstellen Epics auswählen. Außerdem kann der Nutzer auch hier die Aufgabe löschen.



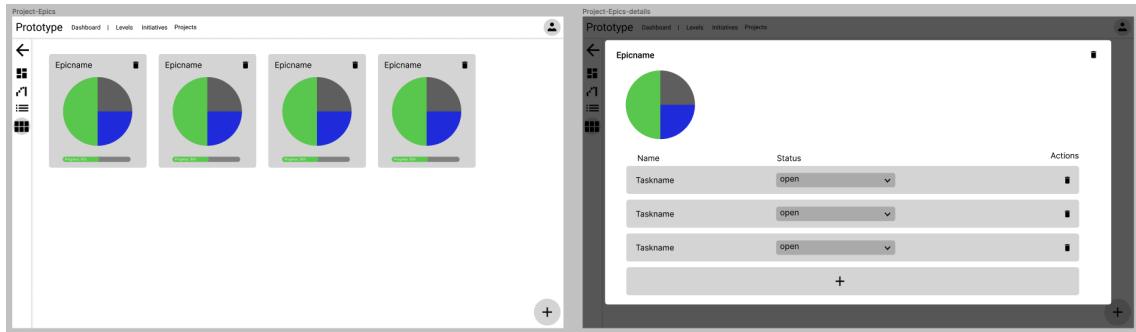
**Abbildung 8:** UX-Prototyp für das Aufgaben-Board

Im Aufgaben-Backlog kann der Nutzer alle Aufgaben des Projekts als Listenansicht sehen. Er kann den Aufgabennamen ändern, den Status jeder Aufgabe mit einem Drop-down-Menü anpassen, Value und Storypoints verändern oder die Aufgabe löschen. Außerdem kann der Nutzer hier eine neue Aufgabe erstellen.



**Abbildung 9:** UX-Prototyp für das Backlog

Befindet sich das Projekt in einer Organisation die Epics verwendet, gibt es ebenfalls die Epic-Übersicht. Hier werden alle Epics in dem Projekt dargestellt und ihr Fortschritt mit einer Grafik visualisiert. Der Nutzer kann neue Epics erstellen und bestehende Löschen. Klickt er auf ein Epic öffnet sich die Epic-Detailansicht in einem Pop-up-Fenster, in dem das Epic umbenannt oder gelöscht werden kann. Außerdem wird eine Liste aller Aufgaben ähnlich wie im Backlog angezeigt, die sich in dem Epic befinden. Hier kann der Nutzer zudem Aufgaben, die noch keinem Epic zugeordnet wurden, auswählen und zu diesem Epic hinzufügen. Aufgaben die sich bereits in dem Epic befinden können gelöscht oder nur aus dem Epic entfernt werden. Erstellt der Nutzer an dieser Stelle eine Aufgabe wird diese automatisch dem Epic hinzugefügt.



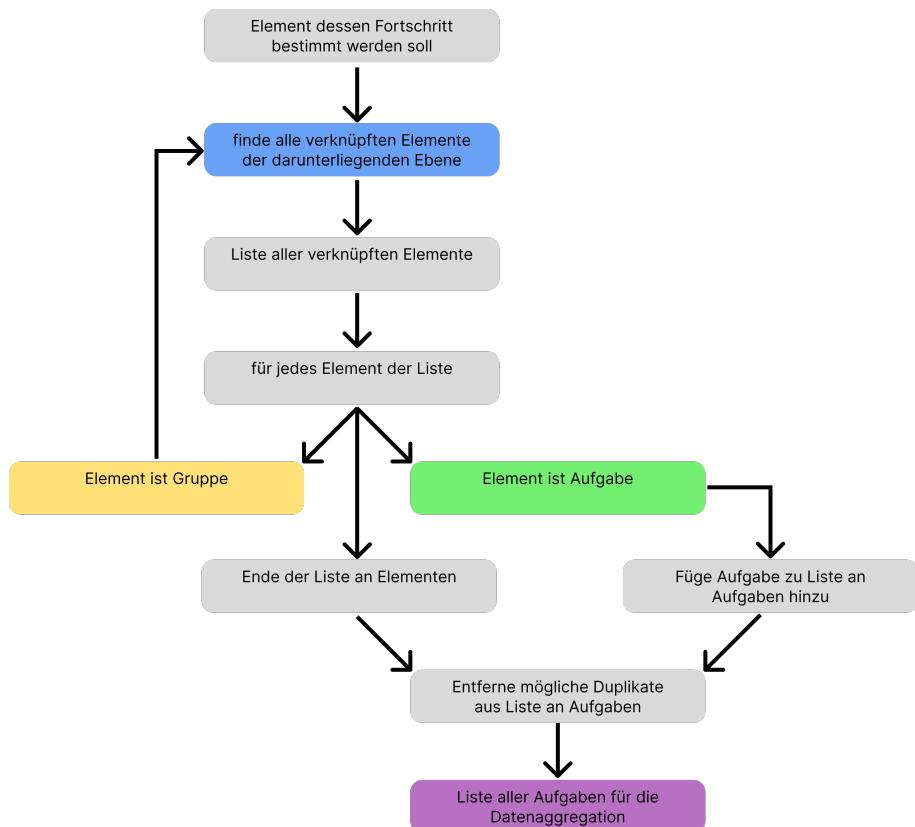
**Abbildung 10:** UX-Prototyp für die Epics-Übersicht

Das Projekt-Dashboard vereint eine Zusammenfassung des Projekt-Fortschritts mit dem Aufgaben-Board und dem Backlog.

### 4.3 Datenaggregation

Für die Fortschrittsaggregation müssen rekursiv Verknüpfungen zu Elementen der Ebene darunter zusammengefasst werden, bis die Elemente Aufgaben sind, deren

Status feststeht. Hierzu beinhalten alle verknüpfbaren Elemente eine Liste mit Referenzen auf verknüpfte Elemente aus der darüber liegenden Ebene. Somit kann überprüft werden welche Elemente der darunterliegenden Ebene mit dem Element, dessen Fortschritt aggregiert werden soll, verknüpft sind. Daraus resultiert eine Liste an Elementen, bei der für jeden Eintrag der Liste genauso wie das für eigentliche Element geprüft werden kann, welche Elemente der darunter liegenden Ebene eine Referenz auf das Element haben. Dies wird so oft wiederholt, bis die verknüpften Elemente Aufgaben sind.



**Abbildung 11:** Prozess für Fortschrittsaggregation

Die Liste an Aufgaben, die sich am Ende des Prozesses ergibt, kann anschließend nach Aufgaben in den verschiedenen Stati sortiert werden. Aus diesen Listen können nun 3 verschiedene Fortschrittsvarianten berechnet werden. Die Anzahl an fertigen Aufgaben geteilt durch die Gesamtmenge an Aufgaben ergibt einen absoluten Fortschritt. Für einen relativen Fortschritt kann die Summe der Storypoints oder des Values aller fertigen Aufgaben durch die Gesamtmenge an Storypoints oder des Values aller Aufgaben geteilt werden, um den relativen Fortschritt zu bestimmen.

## 5 Implementierung

Im folgenden Kapitel wird die Implementierung auf Basis des zuvor erarbeitet Konzepts als webbasierte Anwendung beschrieben und dokumentiert. Dabei steht vor allem die Architektur der einzelnen Teile, aus denen die Anwendung besteht und die Datenstruktur im Vordergrund.

### 5.1 Datenfluss

Der Datenfluss der Anwendung ist ein typischer Datenfluss für Webapplikationen und in Abbildung 12 dargestellt. Die Anwendung besteht aus drei Teilen: Datenbank, Backend und Frontend.



**Abbildung 12:** Datenfluss der Anwendung

Die Datenbank persistiert die Daten der Anwendung. Sie wird durch das Backend verwaltet und gelesen. Das Frontend kann implizit auf diese Daten zugreifen und diese verändern, indem sie diese vom Backend über eine REST-API anfragt. Das Frontend selbst stellt die Daten in einer benutzerfreundlichen Art und Weise dar und ermöglicht es dem Benutzer die Daten zu verändern, indem es als Benutzeroberfläche für die REST-API des Backends fungiert.

Hinterlegt der Nutzer ein Jira-Token kommt eine vierte Komponente hinzu. Diese Komponente ist die JIRA-API, welche die Daten aus Jira zur Verfügung stellt. Um damit den Datenfluss einheitlich zu gestalten, wird die JIRA-API durch das Backend angesprochen, als wäre es eine Datenbank abfrage. Somit hat das Frontend nur eine einzige Quelle an Information.



Abbildung 13: Datenfluss der Anwendung inkl. Jira

## 5.2 Datenstruktur

Die Datenstruktur der Anwendung besteht grundsätzlich aus drei abstrakten und fünf konkreten Klassen. Abstrakte Klassen sind: Entitäten, organisationsbasierende Entitäten und verlinkbare Entitäten. Absolute Klassen sind: Nutzer, Organisation, Level, Gruppe und Aufgabe.

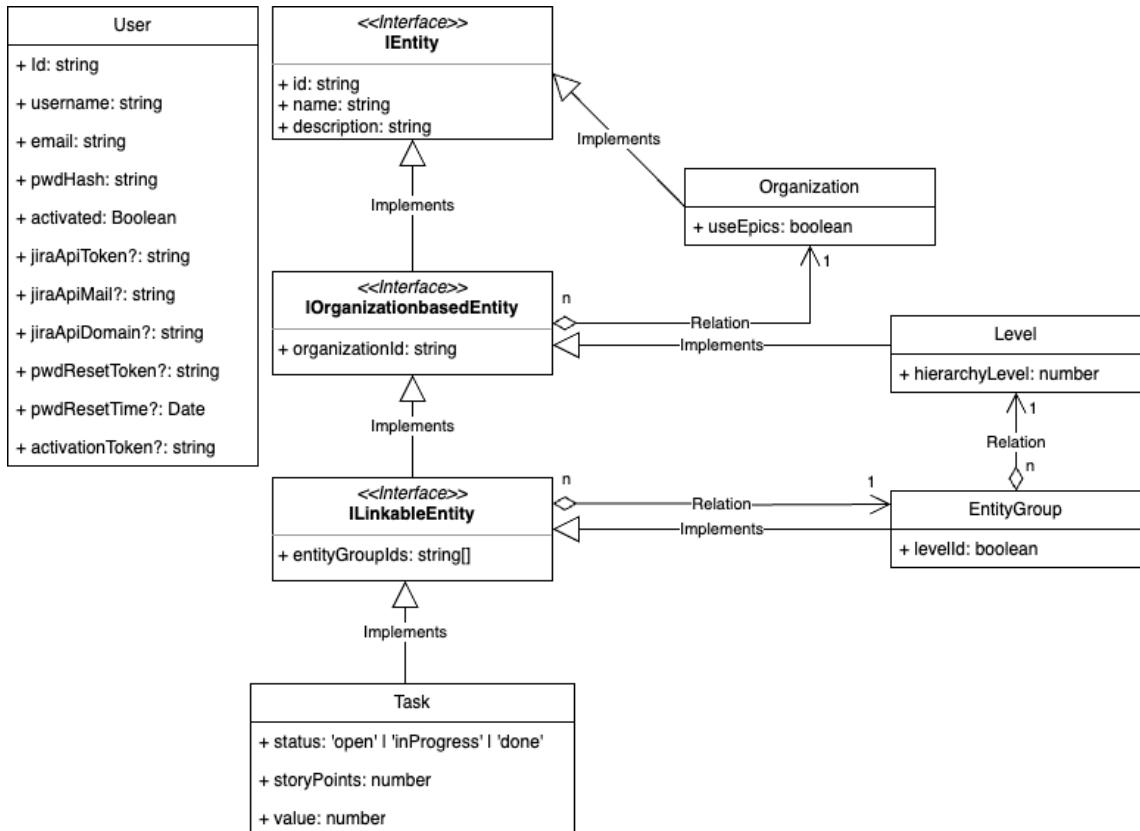


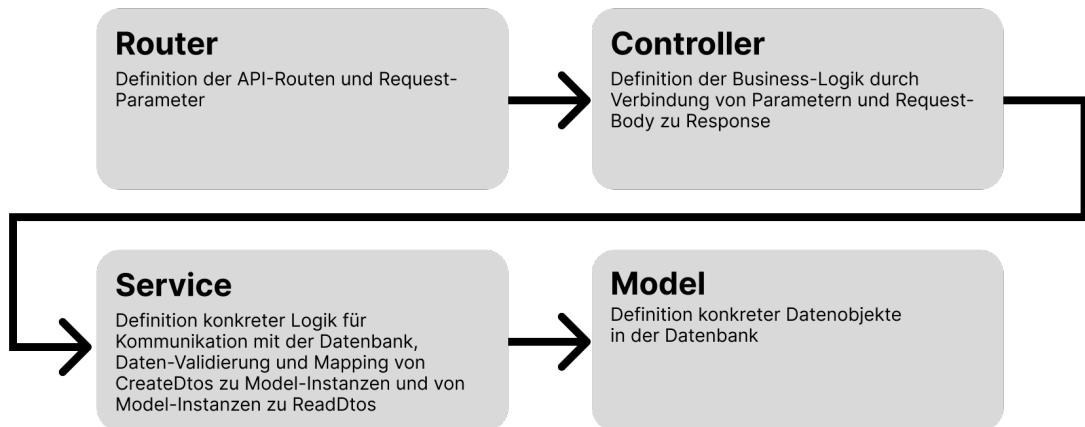
Abbildung 14: UML-Diagramm der Datenstruktur

Jede absolute Klasse, außer die Nutzer, erben von einer oder mehreren abstrakten Klassen. Entitäten sind allgemeine Objekte innerhalb der Anwendung und

stellen die Grundlage der in der Datenbank gespeicherten Datenobjekte dar. Alle Klassen außer Nutzer sind solche Entitäten und implementieren das Interface `IEntity`, welches eine ID zur eindeutigen Identifikation, einen Namen und eine Beschreibung für die Darstellung für den Nutzer beinhaltet. Organisationen und Levels sind direkte Erben dieser Klasse. Die nächste Abstraktionsstufe sind die organisationsbasierenden Entitäten. Diese implementieren zu dem `IEntity` Interface noch `IOrganizationBasedEntity`, welches die ID einer Organisation voraussetzt und die Entität direkt von einer Organisation abhängig macht. Die letzte Abstraktionsstufe sind die verlinkbaren Entitäten. Diese implementieren zu dem `IOrganizationBasedEntity` Interface noch `ILinkableEntity`, welches eine Liste von IDs voraussetzt, mit dem gespeichert wird, mit welchen anderen verlinkbaren Entitäten das Objekt verlinkt ist. Aufgaben und Gruppen sind solche verlinkbare Entitäten.

### 5.3 Backend-Architektur

Das Backend ist eine REST-API, geschrieben mit Node.js und Express in TypeScript und verwendet mongoose als Datenbank-API für MongoDB. Die Architektur beschreibt den Datenfluss mit drei allgemeinen Komponenten: Router, Controller und Service. Der Router bestimmt für einen Request welche Funktion eines Controllers aufgerufen wird. Die aufgerufene Controller-Funktion beinhaltet die Business-Logik, die an den Request gebunden ist und führt diese aus. Um Daten aus der Datenbank zu holen oder die geholten Daten zu modifizieren gibt es für jede Datenklasse einen Service, der die benötigten Datenbankoperationen implementiert und somit von der Business-Logik trennt. Für die Interaktion mit der Datenbank muss außerdem ein sogenanntes Model definiert werden, welches die Beschreibung der Klasse also der Type in TypeScript mit der Datenbank-Collection und den Objekten darin verknüpft.



**Abbildung 15:** Backend Architektur

Für die konkrete Kommunikation mit der REST-API werden zu den konkreten Datenobjekten innerhalb der Datenbank zwei weitere Klassen je Objekt-Klasse definiert. Diese Klassen sind sogenannte Data transfer Objects (DTO). DTOs dienen dazu die Kommunikation zu generalisieren und definieren die Daten, die der Konsument der API durch einen Request erhalten kann und die Daten, die ein Konsument der API zur Verfügung stellen kann, um z. B. ein neues Objekt in der Datenbank zu erstellen. Die zusätzlichen Klassendefinitionen werden durch diese zwei Anwendungsfälle in Read- und Create-/Update-DTOs unterteilt. Wie Create-/Update-DTOs zu einem internen Model gemappt werden und wie aus einem internen Model ein Read-Dto gemappt wird, definiert ebenfalls der zum Model zugehörige Service.

Der Aufbau des Backends gleicht dem Aufbau der Klassen-Abstraktion. Es gibt für jede abstrakte Klasse eine Struktur welche jeweils eine abstrakte Implementierung für Router, Controller, Services und Model beinhalten. Zudem gibt es fünf absolute Strukturen für jede der fünf absoluten Klassen, welche von den verschiedenen abstrakten Strukturen erben.

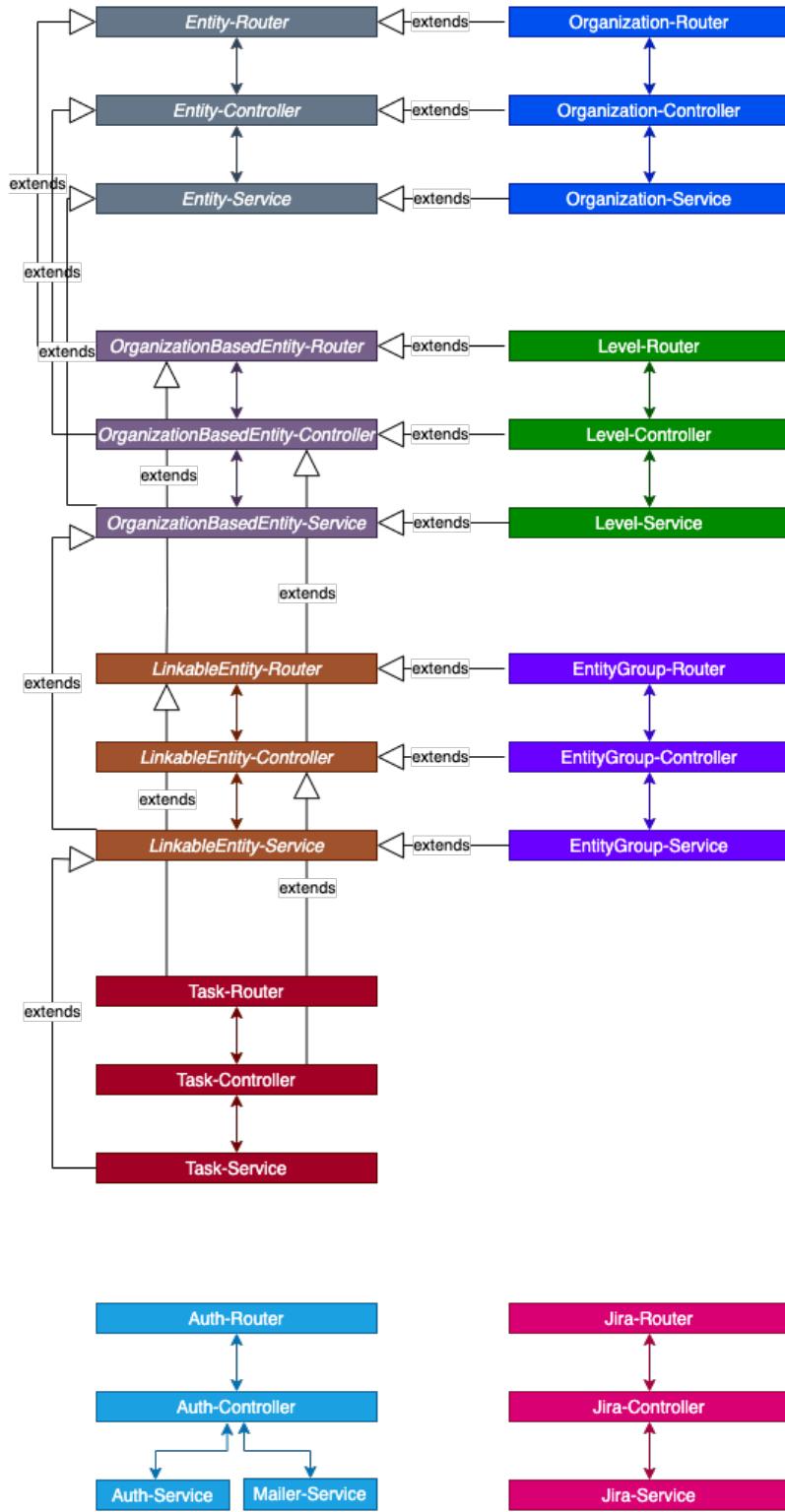


Abbildung 16: Abbildung der BE-Struktur

Für die Authentifizierung werden die Node-Packages `express-sessions` und `passport` verwendet. Zusammen mit einer Persistierung in der Datenbank wird hiermit eine Session-Authentifizierung implementiert, die mit Authentication-Cookies funktioniert.

Eine vollständige Dokumentation der API nach Open-API-Spezifikation-3.0.0 im

Swagger-Format ist hier verfügbar. Dort sind die für den Client zugänglichen Routen inklusive der benötigten Parameter und den zurückgegebenen Daten dokumentiert.

The screenshot shows the Swagger UI interface for the Prototype API. At the top, there's a header with the title "Prototype API" and the OpenAPI logo. Below the header, there's a message about session cookies and a login notice. A dropdown menu for "Servers" is set to "http://localhost:3000/api". On the right, there's an "Authorize" button with a lock icon.

**Auth** (The authorization of the API)

- GET /auth/authorize** authorizes the user and updates the session
- POST /auth/login** logs in the user and creates a session
- POST /auth/logout** logs out the user and deletes the session
- POST /auth/register** created a new user and sends a confirmation email
- POST /auth/activate** activates the user
- POST /auth/forgot-password** sends a password reset email
- POST /auth/reset-password** resets a password with the given password and token

**Group** (API for CRUD operations on entity groups)

- GET /organization/groups/{organizationId}** gets all groups in the organization
- POST /organization/groups/{organizationId}** creates a new group in the organization
- POST /organization/groups/{organizationId}/multiple** creates multiple new groups in the organization
- POST /organization/groups/{organizationId}/link/{entityId}/{entityToLinkId}** links an entity to another entity in another hierarchy
- GET /organization/groups/{organizationId}/{id}** gets group in organization by id
- PUT /organization/groups/{organizationId}/{id}** updates group in organization by id
- DELETE /organization/groups/{organizationId}/{id}** deletes a group and all its related Entities by id

**Level** (API for CRUD operations on levels)

- GET /organization/levels/{organizationId}** gets all levels in the organization
- POST /organization/levels/{organizationId}** creates a new level in the organization
- GET /organization/levels/{organizationId}/{id}** gets level in organization by id
- PUT /organization/levels/{organizationId}/{id}** updates level in organization by id
- DELETE /organization/levels/{organizationId}/{id}** deletes a level and all its related Entities by id

**Organization** (API for CRUD operations on organizations)

- GET /organization** gets all organizations
- POST /organization** creates a new organization
- GET /organization/{id}** gets organization by id
- PUT /organization/{id}** updates organization by id
- DELETE /organization/{id}** deletes an organization and all its related Entities by id

**Task** (API for CRUD operations on tasks)

- GET /organization/tasks/{organizationId}** gets all tasks in the organization
- POST /organization/tasks/{organizationId}** creates a new task in the organization
- POST /organization/tasks/{organizationId}/multiple** creates multiple new tasks in the organization
- POST /organization/tasks/{organizationId}/link/{entityId}/{entityToLinkId}** links an entity to another entity in another hierarchy
- GET /organization/tasks/{organizationId}/{id}** gets task in organization by id
- PUT /organization/tasks/{organizationId}/{id}** updates task in organization by id
- DELETE /organization/tasks/{organizationId}/{id}** deletes a task and all its related Entities by id

**Schemas**

- EntityGroup >
- Level >
- Organization >
- Status >
- Task >
- User >
- Entity >
- Linkable-Entity >
- Organization-Based-Entity >

Abbildung 17: API-Dokumentation

## 5.4 Benutzeroberfläche

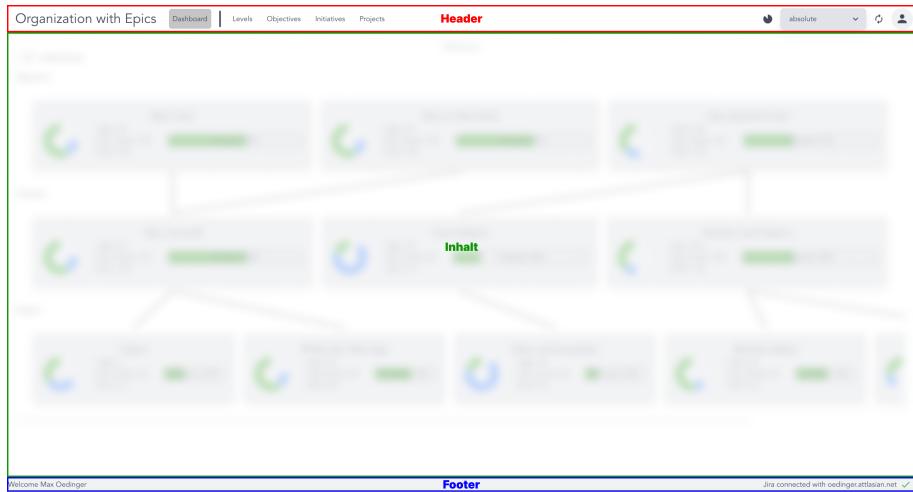
Zur initialen Planung wurde zunächst der UX-Entwurf verwendet, um die grobe Struktur der Benutzeroberfläche zu entwickeln. Durch die spezifische Entwicklung verschiedener Features sind immer wieder Lücken im Entwurf aufgefallen und wurden durch weitere UI-Elemente erweitert, um Funktionalitäten abzudecken, die zuvor nicht innerhalb des Prototyps bedacht wurden. Bis zum fertigen Prototyp haben sich viele der konkreten UI-Elemente verändert, allerdings blieb die Seitenstruktur also welche Informationen auf, welches Seite dargestellt wurden identisch.

Für die Implementierung wurde als Frontend-/UI-Framework Vue.js verwendet. Vue.js ist eine Framework für die Entwicklung von Single-Page-Webanwendungen. Es ist ein JavaScript-Framework, welches auf dem Model-View-ViewModel (MVVM) basiert. Die UI wurde also in verschiedene Komponenten zerteilt. Die Komponenten werden in zwei Kategorien unterteilt: Views und Components. Views sind logisch voneinander getrennte Seiten der Anwendung, während Components kleinere Bestandteile der Views zur Vereinfachung der in der View benötigten Logik oder wiederverwendbare UI-Elemente sind. Zur weiteren Strukturierung werden hier noch sogenannte Layouts verwendet. Layouts stellen die Grundstruktur der Anwendung dar, welche von mehreren Views verwendet wird. Für das Styling der Anwendung wurde das CSS-Framework Tailwind-CSS verwendet. Dabei handelt es sich aber um kein Design-Framework, sondern lediglich um eine alternative Art natives CSS zu verwenden. Dementsprechend hat das Frontend sein eigenes Design, welches sich an keine Festen Design-Vorgaben eines anderen Systems hält.

### 5.4.1 Layouts

Die Anwendung teilt sich zunächst in zwei solcher Layouts: Authentifizierung und eigentliche Anwendung.

Das Layout der Authentifizierung beschreibt nur die Positionierung der relevanten Elemente in der Mitte des Bildschirms, da sich alle Seiten der Authentifizierung diese Eigenschaft teilen.



**Abbildung 18:** Visualisierung des Layouts

Das Layout der eigentlichen Seite teilt die Seite in drei Teile, den Header, den Inhalt und einen Footer. Der Header beinhaltet die Navigationsleiste, die den Nutzer durch die Anwendung führt und oben rechts ein Aktionsmenü mit dem der Nutzer sich jederzeit ausloggen kann oder in die Einstellungen bzw. sein Profil navigieren kann. Zudem kann der Nutzer links neben dem Aktionsmenü die Daten der Anwendung erneut laden und die Fortschrittsmessung zwischen Absolut, Storypoints und Value auswählen. Der Inhalt ist der Bereich in dem die verschiedenen Views dargestellt werden. Im Footer sind Informationen über den aktuell eingeloggten Nutzer und seine Verbindung zu Jira enthalten.

#### 5.4.2 Views und Components

Die Views beinhalten die im UX-Entwurf beschriebenen Anwendungsteile: Login, Registrierung, Passwort vergessen, Passwort Reset, Dashboard, Einstellungen, Profil, Organisationen, Level-Übersicht, Gruppenansicht, Gruppendetailansicht, Projektübersicht, Projekt-Dashboard, Aufgaben-Board, Aufgaben-Backlog und Epic-Übersicht. Außerdem gibt es eine Not-Found-View welche dargestellt wird, wenn der Nutzer versucht eine nicht existierende Seite aufzurufen und eine Home-View, die als Homepage der Anwendung dient. Hier kann der Nutzer das Dokument der Bachelorarbeit sehen und die API-Dokumentation aufrufen.

Die Components enthalten mehrfach verwendetet UI-Elemente, wie z. B. Buttons, Icons, Textfelder, Dropdowns, Drag-And-Drop-Elemente, Dialogfenster, etc.

### 5.4.3 Datenverwaltung

Für die anwendungsübergreifende Datenverwaltung wird das Framework Pinia als Store verwendet. Der Store dient dazu die Daten nicht Kontextspezifisch zu speichern, sondern in der gesamten Anwendung verfügbar zu machen. Somit müssen die Daten nicht nach jeden Seitenwechsel neu geladen werden, sondern können aus dem Store abgerufen werden. Dies verbessert die Performance der Anwendung. Außerdem werden alle Kommunikationen mit dem Backend über den Store abgewickelt und durch die Ergebnisse der Kommunikationen werden die Daten im Store aktualisiert. Daten, die aus dem Store abgerufen werden, sind reactive, das bedeutet, dass Änderungen der Daten ebenfalls in den Komponenten, welche die Daten verwenden, reflektiert werden. Ein Store besteht aus drei verschiedenen Teilen: State, Getters und Actions. Der State beschreibt den Zustand der Daten. Getters sind Funktionen, die Daten aus dem State für verschiedene Anwendungsfälle transformiert oder gefiltert zurückgeben. Actions sind Funktionen, welche die Kommunikation mit dem Backend abwickeln und mit den zurückgegebenen Daten den State mutieren.

Es gibt mehrere Stores, die verschiedene Daten speichern. Für allgemeine Informationen gibt es den AppStore, welcher speichert welche Fortschrittsmessung ausgewählt wurde und ob der Light- oder Darkmode ausgewählt wurde. Der AppStore wird zusätzlich im Local-Storage des Browsers persistiert, was dafür sorgt, dass diese Informationen auch bei einem Neuladen der Seite erhalten bleiben.

Im AuthStore wird der eingeloggte Nutzer gespeichert und ob der Nutzer eingeloggt ist und Login und Logout definiert. Für Daten, die in der Backend- und Datenstruktur als Entitäten bezeichnet werden gibt es auch hier eine Abstraktion der Stores für die Verwaltung dieser Entitäten. Es gibt also auch hier eine Abstraktion in EntityStore, OrganizationBasedEntity und LinkableEntity, von welchen die tatsächlich verwendeten Stores für die verschiedenen Datentypen erben. Anders als die Backendstruktur funktioniert die Vererbung hier aber nicht über Klassen, da Pinia einem funktionalen Implementierungsschema folgt. Also gibt es MakeFunktionen für jedes der drei verschiedenen Teile des Stores, welche mit Generischen-Type-Parametern State, Getter und Actions mit den richtigen Typen erzeugt. Für simple Stores wie z. B. den OrganizationStore beschränkt sich die absolute Implementierung dann wie folgt:

```
1 export const useOrganizationStore: EntityStoreDefinition <
```

```

2   IOrganization ,
3   'organization'
4 > = defineStore('organization', {
5   state: makeEntityState<IOrganization>(organizationService),
6   getters: makeEntityGetters<IOrganization>(),
7   actions: makeEntityActions<IOrganization>(),
8 });

```

**Listing 1:** Implementierung des OrganizationStores

Für Stores mit zusätzlichen Getters, werden die MakeFunktionen erweitert, wie beispielsweise im LevelStore:

```

1 interface LevelGetters extends OrganizationBasedEntityGetters<
2   ILevel > {
3   getNextHierarchyLevel(): number;
4   isProjectLevel(): (levelId: string) => boolean;
5   getLowerLevel(): Level | undefined;
6 }
7
8 type LevelStore = OrganizationBasedEntityStore<
9   ILevel ,
10  'level' ,
11  OrganizationBasedEntityState<ILevel>,
12  LevelGetters
13 >;
14 type LevelStoreDefinition = OrganizationBasedEntityStoreDefinition<
15  ILevel ,
16  'level' ,
17  OrganizationBasedEntityState<ILevel>,
18  LevelGetters
19 >;
20 const levelStore: PiniaStore<LevelStore> = {
21   state: makeOrganizationBasedEntityState<ILevel>(levelService),
22   getters: {
23     ...makeOrganizationBasedEntityGetters<ILevel>(),
24     getNextHierarchyLevel(state) {
25       const currentOrganization = useOrganizationStore().
26       currentEntity;
27       if (!currentOrganization) return 0;
28       return state.entities.filter(

```

```

28         ({ organizationId }) => organizationId ===
29         currentOrganization.id
30         ).length;
31     },
32     isProjectLevel(state) {
33         return (levelId: string) => {
34             const projectLevel = useOrganizationStore().
35             currentEntity?.useEpics
36             ? 1
37             : 0;
38             return (
39                 state.entities.find((level) => level.id ===
40                 levelId)
41                     ?.hierarchyLevel === projectLevel
42             );
43         },
44         getLowerLevel(state) {
45             const currentLevel: Entity<OrganizationBasedEntity<
46             ILevel>> =
47                 state.currentEntity;
48             if (!currentLevel) return;
49
50             return state.entities.find(
51                 (x) =>
52                     x.organizationId === currentLevel.
53                     organizationId &&
54                     x.hierarchyLevel === currentLevel.
55                     hierarchyLevel - 1
56             );
57         },
58     },
59     actions: makeOrganizationBasedEntityActions<ILevel>(),
60 };
61
62 export const useLevelStore: LevelStoreDefinition = defineStore(
63     'level',
64     levelStore
65 );

```

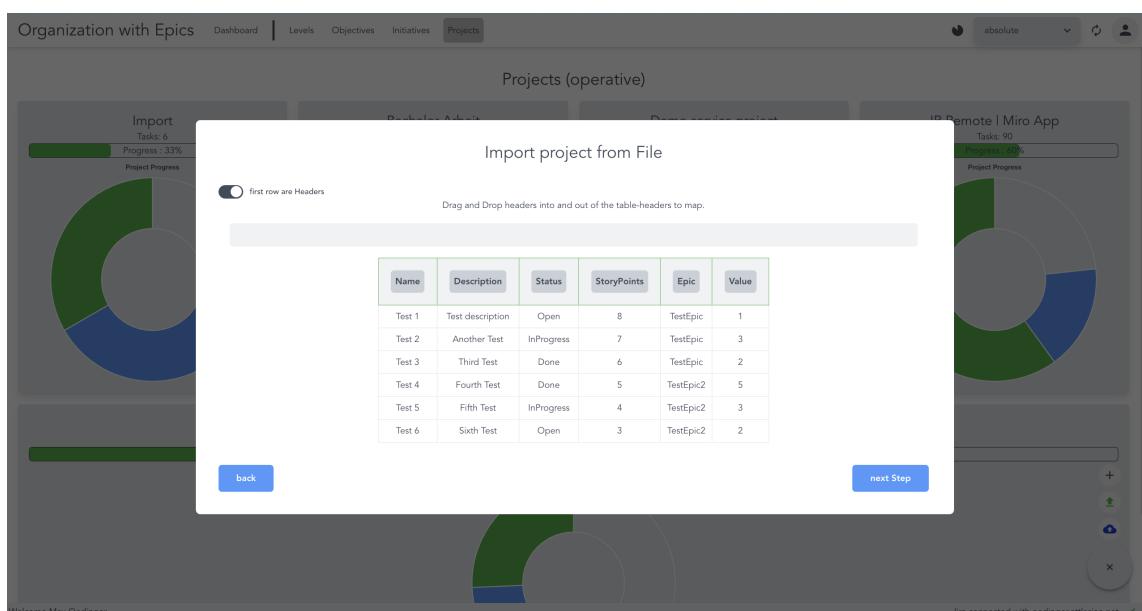
**Listing 2:** Implementierung des LevelStores

Stores für Entitäten, haben in ihrem State zusätzlich ein Loaded Attribut in dem gespeichert wird, ob die Daten bereits von Server abgefragt wurde, bzw. wenn es sich um eine OrganizationBasedEntity handelt, ob die Daten für diese Organisation bereits geladen wurde, um mehrfache Requests für die gleichen Daten zu vermeiden und damit die Performance zu verbessern.

Der Jira-Store speichert alle den Nutzer zugänglichen Projekte und bietet mit seinen Actions die Möglichkeit die relevanten Daten zu diesem Projekt zu holen, wenn der Nutzer den Import-Prozess für ein Jira-Projekt startet.

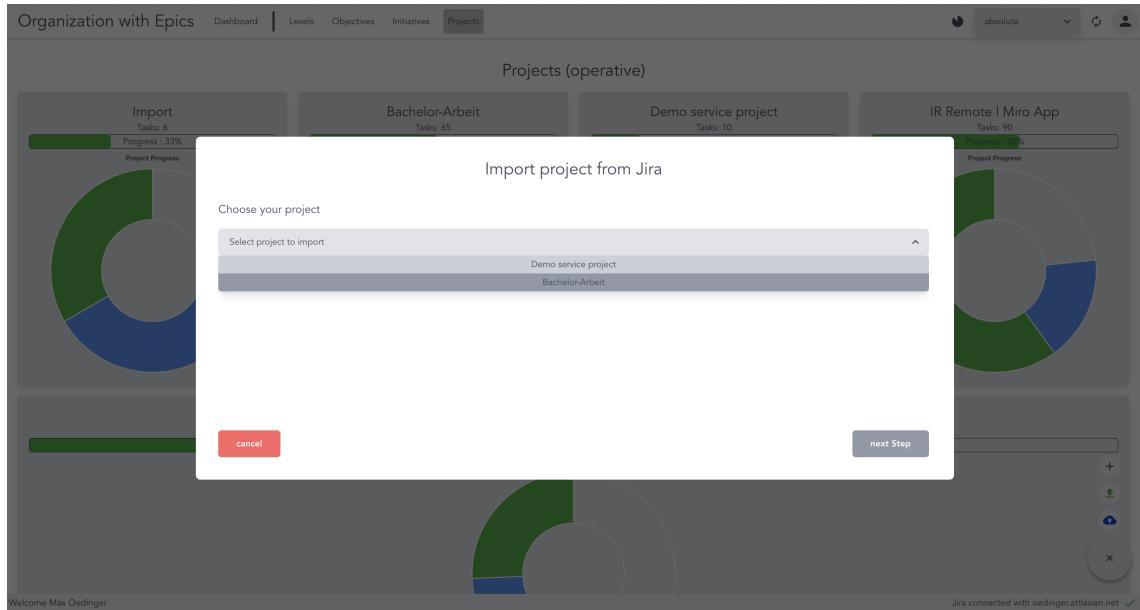
## 5.5 Projekt-Import

Um die Verwendung der Anwendung insbesondere für Projekte, welche eine große Menge an Daten beinhalten können und meist in externen Tools dokumentiert und verwaltet werden, zu vereinfachen, wurde ein Import-Prozess sowohl für Projektdaten in Excel-Dateien, als auch für Jira-Projekte implementiert. Dieser Import Prozess erlaubt ein individuelles Mapping der Daten aus den externen Tools auf die Datenstruktur der Anwendung. Das Mapping bei einer Excel-Tabelle erwartet eine Excel-Tabelle, welche alle Aufgaben eines Projekts enthält. Hierbei kann der Nutzer die Felder aus der Anwendung mit Drag-and-Drop Spalten in der Tabelle zuordnen. Sollten Spalten bereits genauso wie Felder innerhalb der Anwendung heißen, werden diese Felder automatisch gemapped, können vom Nutzer aber auch wieder abgeändert werden.



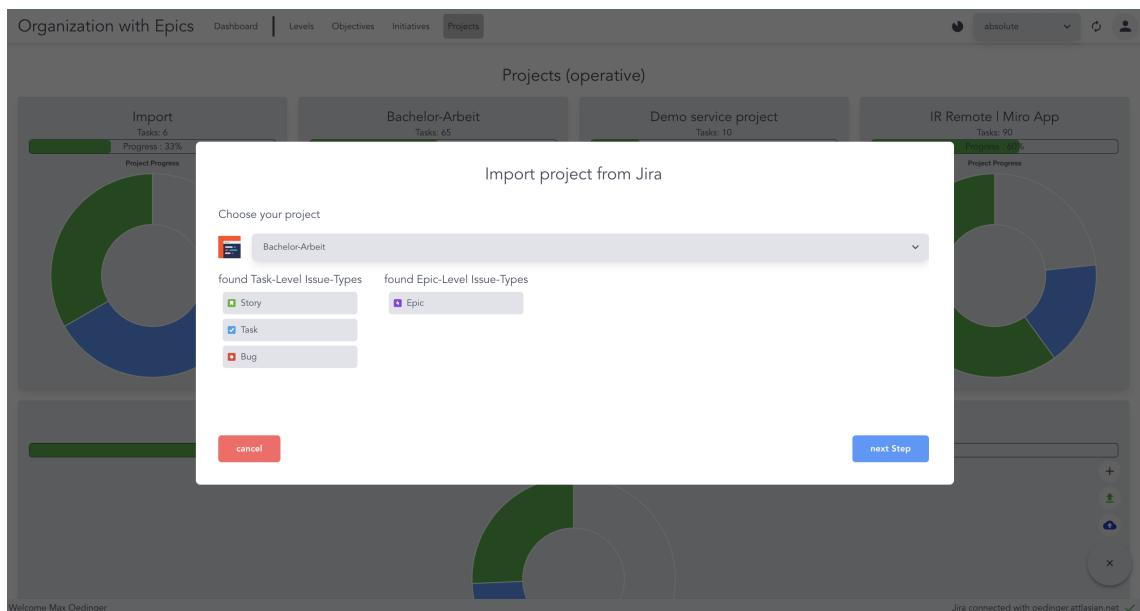
**Abbildung 19:** Import Feld Mapping

Anschließend wird für das Feld welches den Status der Aufgabe enthält einem Status innerhalb der Anwendung zuordnen, enthält das Feld für eine Aufgabe den gleichen Wert wie der Zugeordnete Status, wird dieser verwendet.



**Abbildung 20:** Import Jira Projekt Auswahl

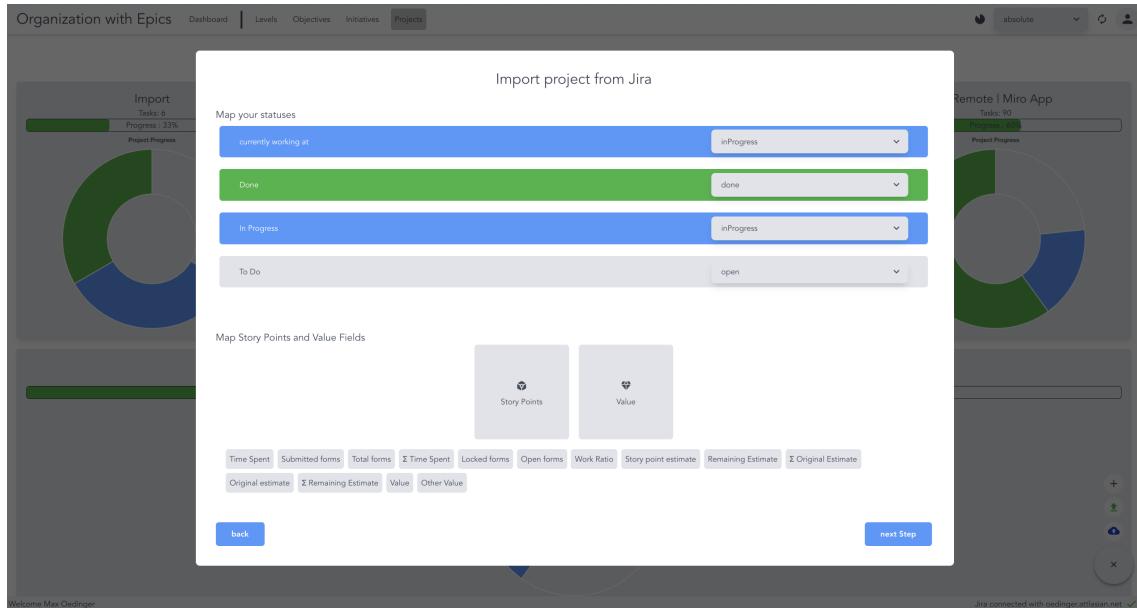
Importiert der Nutzer von Jira kann er zunächst ein Projekt auswählen.



**Abbildung 21:** Import Jira Ansicht nach Projekt Auswahl

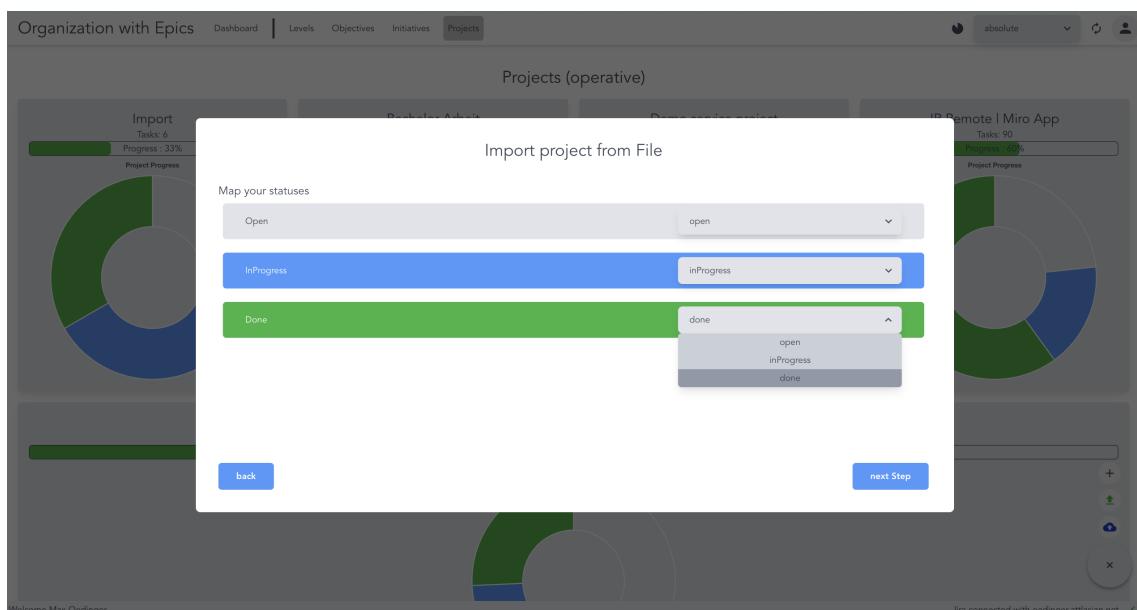
Für Jira-Projekte kann das Mapping der Felder größtenteils automatisch durchgeführt werden, das bedeutet, dass Aufgaben und Epics sofort erkannt werden. Die

Stati der Aufgaben können anhand der Metadaten von JIRA auch automatisch aus-gelesen werden, der Nutzer kann dennoch Für jeden Status aus JIRA ändern, welcher Status im Prototypen zugeordnet wird, falls die tatsächliche Verwendung von den Meta-Daten abweichen sollte.



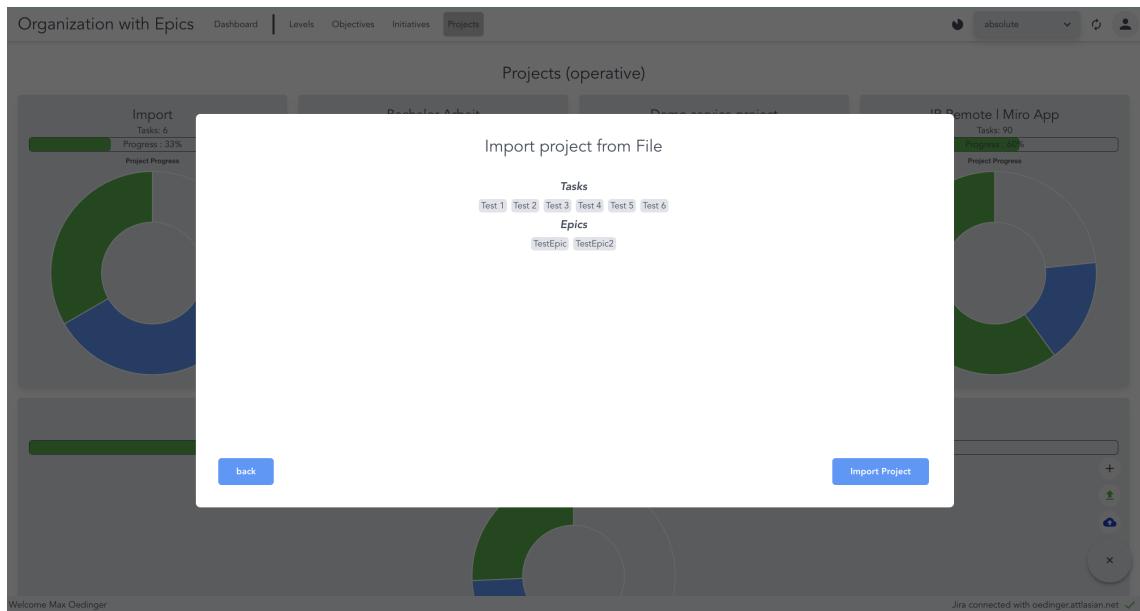
**Abbildung 22:** Import Jira Mapping

Außerdem kann der Nutzer an dieser Stelle Custom-Fields für Issues aus Jira, welche Zahlenwerte enthalten auf Value und/oder Storypoints mappen.

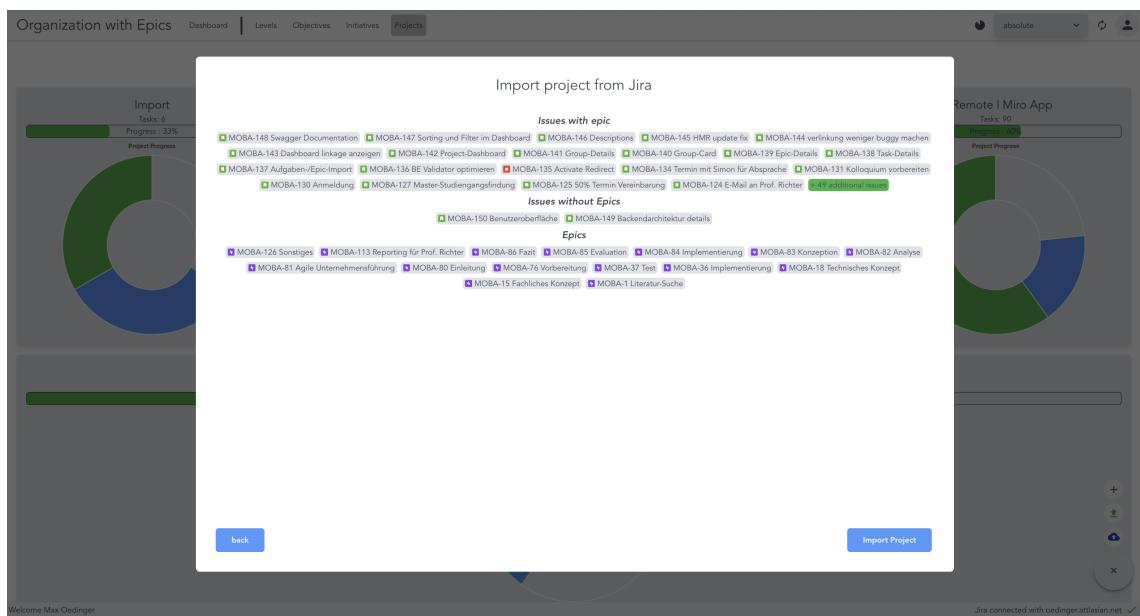


**Abbildung 23:** Import Status Mapping

Zuletzt erhält der Nutzer noch einmal eine Übersicht über die importierten Daten und kann den Import-Prozess starten.



**Abbildung 24:** Import Übersicht bei Excel Datei

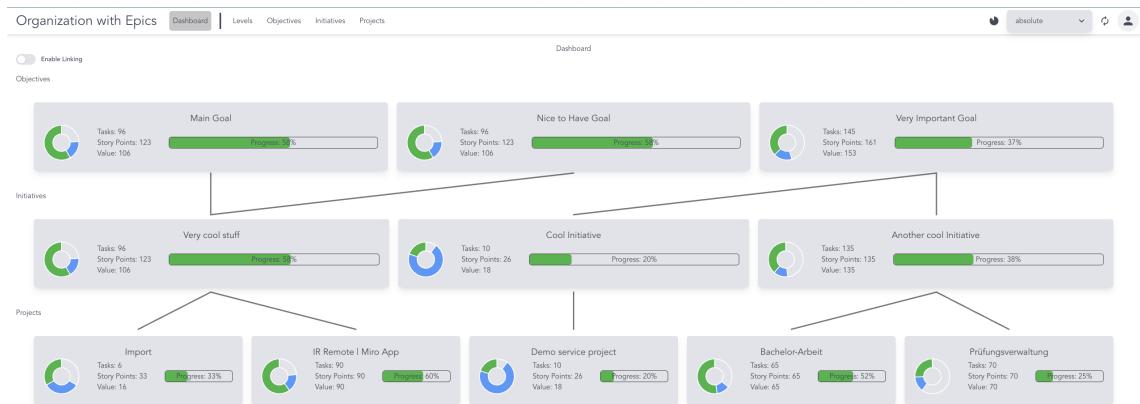


**Abbildung 25:** Import Übersicht bei Jira Import

## 5.6 Visualisierung/Datendarstellung

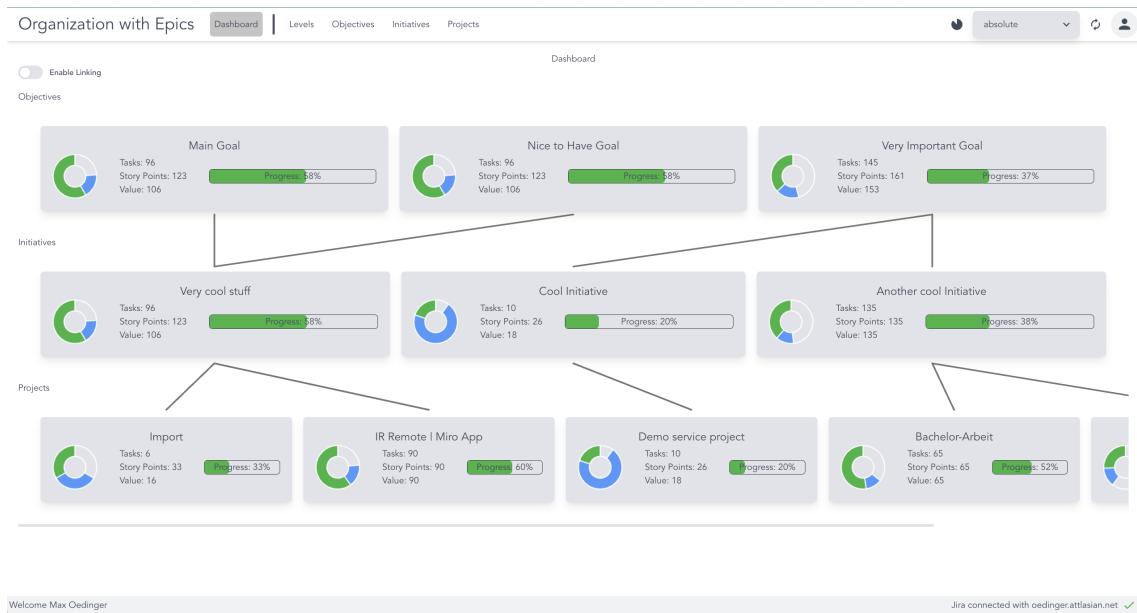
Für die generelle Visualisierung der gesamten Daten einer Organisation dient das Dashboard, welches den komplexesten Teil der Anwendung darstellt.

Hier werden alle Gruppen der Organisation hierarchisch sortiert dargestellt und ihre Verknüpfungen untereinander visualisiert. Um die Übersicht bei vielen Gruppen beizubehalten werden die Gruppen innerhalb einer hierarchischen Ebene nach ihren Verbindungen zu den darüberliegenden Gruppen sortiert, sodass Gruppen, die mit der gleichen Gruppe in der Ebene darüber verknüpft sind, nebeneinander dargestellt werden. Die Verbindungslinien werden somit möglichst kurz und übersichtlich dargestellt.



**Abbildung 26:** Dashboard

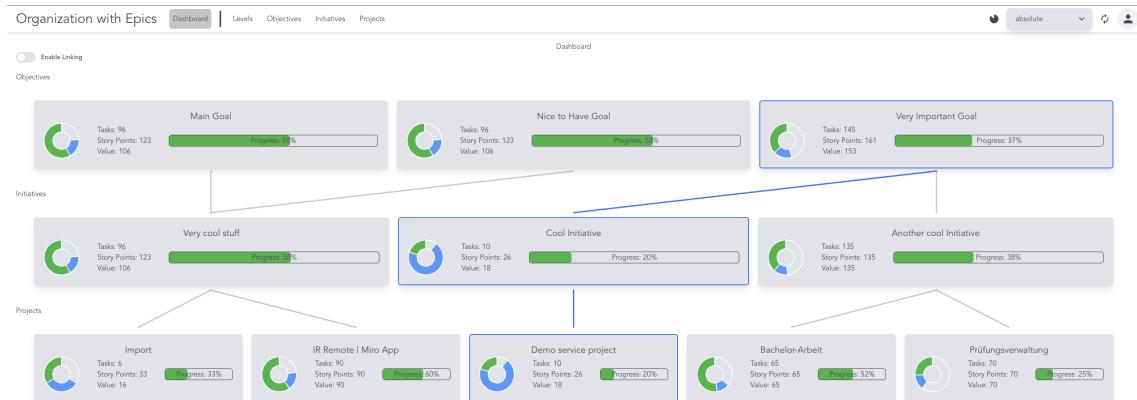
Um die Verbindungslinien darzustellen wurde ein SVG-Element im Hintergrund der Seite verwendet, welches die Verbindungslinien unabhängig von anderen Elementen auf der Seite darstellen kann. Die Start- und Endkoordinaten der Linien werden durch Positionen der Gruppen innerhalb der Seite berechnet. Dabei muss zusätzlich beachtet werden, wenn ein Level so viele Gruppen beinhaltet, dass sich einige Gruppen außerhalb des sichtbaren Bereichs befinden und durch einen Scroll in den sichtbaren Bereich bewegt werden können. In diesem Fall müssen die Koordinaten der Linien entsprechend der Scroll-Position der verschobenen Gruppen neu berechnet werden.



**Abbildung 27:** Dashboard mit scrollbaren Gruppen

Der Nutzer hat die Möglichkeit Verlinkung über ein Toggle zu aktivieren, wodurch über jeder Gruppe, die verlinkt werden kann, ein Punkt erscheint, den der Nutzer mit Drag-and-Drop verwenden kann, um die Gruppe mit einer anderen Gruppe zu verbinden. Anhand der Start-Position und aktuellen Position des Mauszeigers wird währenddessen eine neue Linie dargestellt, die dem Nutzer zeigt, wie die Verbindung aussieht die er erzeugt. Anschließend werden die Verbindungslien anhand der neuen Positionen der Gruppen erneut berechnet.

Zusätzlich zu den Linien kann der Nutzer den Mauszeiger über eine Gruppe bewegen, wodurch die Gruppe und die Gruppen, die mit dieser Gruppe verknüpft sind, inklusive aller relevanten Verbindungslien, blau hervorgehoben werden.



**Abbildung 28:** Dashboard mit hervorgehobenen Gruppen

Für die Erzeugung der Doughnut-Diagramme wird Chart.js verwendet, da es eine simple und visuell ansprechende Möglichkeit bietet einfache Diagramme zu erstellen. Durch das Plugin Vue-Chartjs gibt es die Möglichkeit die Diagramme in Vue.js direkt als Komponente einzubinden, was die Verwendung weiter erleichtert.

## 5.7 lokale Inbetriebnahme

Um die Anwendung zu starten werden zwei Möglichkeiten angeboten. Die erste Möglichkeit besteht in der Verwendung von Docker-Compose. Die zweite Möglichkeit besteht in der Verwendung von Node.js, Vite, und Docker für die MongoDB. Detaillierte Beschreibungen für die konkreten Details der Inbetriebnahme sind in der `README.md`-Datei im `prototype`-Ordner des Repositories zu finden.

## 5.8 CI/CD

Für eine schnelle und einfache Möglichkeit die Anwendung als produktive Anwendung zu testen und Verbesserungen zu deployen wurde eine CI/CD-Pipeline mit GitHub-Actions eingerichtet. Die GitHub-Action definiert, wann die Pipeline ausgeführt werden soll und welche Schritte in der Pipeline ausgeführt werden sollen. Der Trigger für die Pipeline ist hier das Mergen und daraus resultierenden Schließen eines Pullrequests der als Ziel den main-Branch des Repositories hat.

```

1 name: Docker Image CI/CD Pipeline
2
3 on:
4   pull_request:
5     types:
6       - closed
7     branches:
8       - main
9
10 jobs:
11   build:
12     if: github.event.pull_request.merged == true
13     runs-on: self-hosted
14     environment:
15       name: production
16     env:
17       MONGO_PASSWORD: ${{ secrets.MONGO_PASSWORD }}
```

```

18     SECRET: ${{ secrets.SECRET }}
19     CLIENT_APP_URL: ${{ vars.CLIENT_APP_URL }}
20     MONGO_DB: ${{ vars.MONGO_DB }}
21     MONGO_URL: ${{ vars.MONGO_URL }}
22     MONGO_USER: ${{ vars.MONGO_USER }}
23     SMTP_HOST: ${{ vars.SMTP_HOST }}
24     SMTP_PORT: ${{ vars.SMTP_PORT }}
25     SMTP_USER: ${{ vars.SMTP_USER }}
26     VITE_API_URL: ${{ vars.VITE_API_URL }}
27     VITE_APP_DEV_MODE: ${{ vars.VITE_APP_DEV_MODE }}

28 steps:
29   - uses: actions/checkout@v3
30   - name: Build the Docker image
31     run:
32       cd prototype &&
33       docker build . --file "Dockerfile" --tag bachelor:latest
34       --build-arg MONGO_PASSWORD=${{ env.MONGO_PASSWORD }}
35       --build-arg SECRET=${{ env.SECRET }}
36       --build-arg CLIENT_APP_URL=${{ env.CLIENT_APP_URL }}
37       --build-arg MONGO_DB=${{ env.MONGO_DB }}
38       --build-arg MONGO_URL=${{ env.MONGO_URL }}
39       --build-arg MONGO_USER=${{ env.MONGO_USER }}
40       --build-arg SMTP_HOST=${{ env.SMTP_HOST }}
41       --build-arg SMTP_PORT=${{ env.SMTP_PORT }}
42       --build-arg SMTP_USER=${{ env.SMTP_USER }}
43       --build-arg SMTP_PASS=${{ env.SMTP_PASS }}
44       --build-arg VITE_API_URL=${{ env.VITE_API_URL }}
45       --build-arg VITE_DEV_MODE=${{ env.VITE_APP_DEV_MODE }}

46 deploy:
47   needs: build
48   runs-on: self-hosted
49   environment:
50     name: production
51   env:
52     MONGO_PASSWORD: ${{ secrets.MONGO_PASSWORD }}
53     SECRET: ${{ secrets.SECRET }}
54     CLIENT_APP_URL: ${{ vars.CLIENT_APP_URL }}
55     MONGO_DB: ${{ vars.MONGO_DB }}
56     MONGO_URL: ${{ vars.MONGO_URL }}
57     MONGO_USER: ${{ vars.MONGO_USER }}
```

```

58     SMTP_HOST: ${{ vars.SMTP_HOST }}
59     SMTP_PORT: ${{ vars.SMTP_PORT }}
60     SMTP_USER: ${{ vars.SMTP_USER }}
61     SMTP_PASS: ${{ secrets.SMTP_PASS }}
62   steps:
63     - name: Deploy the newest Docker image
64       run: docker compose -f "prototype/docker-compose.ci.yml" up
65         -d
66     - name: Delete all old and unused Images
67       run: docker system prune -af

```

**Listing 3:** GitHub-Action

Die Pipeline besteht aus zwei Teilen, sogenannten Jobs: Build und Deploy. Build erzeugt aus dem geupdateten Quellcode ein neues Docker-Image nach den konkreten Anweisungen die in dem Dockerfile im root-Verzeichnis des Prototypen definiert sind. Bei dem Dockerfile handelt es sich um ein multi-stage Dockerfile, welches in drei Schritten ein Image erzeugt. Diese drei Schritte sind: frontend-build, backend-build, und production.

```

1 # build stage
2 # frontend
3 FROM node:16-alpine as fe-build-stage
4 WORKDIR /app
5 ARG VITE_API_URL
6 ARG VITE_DEV_MODE
7 COPY ./frontend/package*.json .
8 RUN npm ci
9 COPY ./frontend/ .
10 RUN npm run build:ci
11
12 # backend
13 FROM node:16-alpine as be-build-stage
14 WORKDIR /app
15 COPY ./backend/package*.json .
16 RUN npm ci
17 COPY ./backend .
18 RUN npm run build
19 RUN mkdir ./build/views
20 RUN mkdir ./build/views/frontend
21 COPY --from=fe-build-stage /app/dist ./build/views/frontend

```

```

22
23
24 # production stage
25 FROM node:16-alpine as production-stage
26 WORKDIR /app
27 COPY ./backend/package*.json ./
28 RUN npm ci
29 COPY --from=be-build-stage /app/build /app
30 EXPOSE 3000
31 CMD [ "node", "./main.js"]

```

**Listing 4:** Dockerfile zum Bauen des Images

Im ersten Schritt wird der Frontend-Code kompiliert. Im zweiten Schritt wird der Backend-Code kompiliert und das kompilierte Frontend aus dem ersten Schritt zur statischen Auslieferung in einen bestimmten Ordner kopiert. Im dritten Schritt wird ein Node-Image für die Produktivumgebung erzeugt, welches das kompilierte Backend inklusive des Frontends enthält und mit einem Start-Befehl der Webserver gestartet wird. Das resultierende Image, welches nun eine lauffähige Version der Anwendung enthält, wird anschließend als neueste Version des Images getagt. Deploy führt ein Update der konkreten Produktivumgebung durch. Dazu wird das Image, welches im Build-Job erzeugt wurde, und das laufende Image ersetzt. Hierzu wird die Docker-Compose-Datei verwendet, welche die Konfiguration der Docker-Services der Produktivumgebung beschreibt. Diese Datei referenziert immer die neueste Version des gebauten Images und einen Reverse-Proxy-Service von "Traefik", welcher dazu dient das Portmapping auf dem Server zu übernehmen und mithilfe von Letsencrypt ein SSL-Zertifikat für die angegebene Domain auszustellen. Zuletzt werden alle alten und unbenutzt Images gelöscht.

## 6 Evaluation

### 6.1 Praxistest

Für den Praxistest wurden mit den Experten, die für die Evaluation herangezogen wurden, praxisnahe Daten erstellt, die ein fiktives Unternehmen darstellen sollen. Anhand der Daten im Prototypen soll, beurteilt werden, wie praxisrelevant die Funktionalitäten der Software sind und welche Verbesserungen und/oder Funktionalitäten für eine produktive Nutzung benötigt werden.

### 6.2 Optimierungsvorschläge

- Jira-Sync
- Velocity und andere zeitbasierte Metriken
- mehr Filterfunktionalitäten
- Rollen und Lesebeschränkungen

## 7 Fazit

### 7.1 Ergebnis

### 7.2 Reflexion

### 7.3 Ausblick

## Literatur

- [1] Daniel J. Fernandez und John D. Fernandez. „Agile Project Management —Agilism versus Traditional Approaches“. In: *Journal of Computer Information Systems* 49.2 (2008), S. 10–17. DOI: 10.1080/08874417.2009.11646044. eprint: <https://www.tandfonline.com/doi/pdf/10.1080/08874417.2009.11646044>. URL: <https://www.tandfonline.com/doi/abs/10.1080/08874417.2009.11646044>.
- [2] Taghi Javdani Gandomani u. a. „The Role of Project Manager in Agile Software Teams: A Systematic Literature Review“. In: *IEEE Access* 8 (2020), S. 117109–117121. DOI: 10.1109/ACCESS.2020.3004450.
- [3] H. Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. 2013.
- [4] Pedro Serrador und Rodney Turner. „The Relationship between Project Success and Project Efficiency“. In: *Project Management Journal* 46.1 (2015), S. 30–39. DOI: 10.1002/pmj.21468. eprint: <https://doi.org/10.1002/pmj.21468>. URL: <https://doi.org/10.1002/pmj.21468>.
- [5] Lehtineva Lassi Agbejule Adebayo. „The relationship between traditional project management, agile project management and teamwork quality on project success“. In: *International Journal of Organizational Analysis* 30.7 (2022), S. 124–136. DOI: 10.1108/IJOA-02-2022-3149. URL: <https://doi.org/10.1108/IJOA-02-2022-3149>.
- [6] Ursula Kusay-Merkle. *Agiles Projektmanagement im Berufsalltag - Für mittlere und kleine Projekte*. Deutschland: © Springer-Verlag GmbH, ein Teil von Springer Nature, 2018, S. 171–173.
- [7] Ursula Kusay-Merkle. *Agiles Projektmanagement im Berufsalltag - Für mittlere und kleine Projekte*. Deutschland: © Springer-Verlag GmbH, ein Teil von Springer Nature, 2018, S. 149–152.
- [8] Na Mi Nguyen u. a. „The use of effectuation in projects: The influence of business case control, portfolio monitoring intensity and project innovativeness“. In: *International Journal of Project Management* 36.8 (2018), S. 1054–1067. ISSN: 0263-7863. DOI: <https://doi.org/10.1016/j.ijproman.2018.07.005>.

- 08 . 005. URL: <https://www.sciencedirect.com/science/article/pii/S0263786318301625>.
- [9] Miia Martinsuo und Päivi Lehtonen. „Role of single-project management in achieving portfolio management efficiency“. In: *International Journal of Project Management* 25.1 (2007), S. 56–65. ISSN: 0263-7863. DOI: <https://doi.org/10.1016/j.ijproman.2006.04.002>. URL: <https://www.sciencedirect.com/science/article/pii/S026378630600069X>.
  - [10] Elkin Doney Suárez Gómez. „Scalable agile frameworks in large enterprise project portfolio management“. Diss. Pontificia Universidad Católica del Perú, 2022, S. 14.
  - [11] Elkin Doney Suárez Gómez. „Scalable agile frameworks in large enterprise project portfolio management“. Diss. Pontificia Universidad Católica del Perú, 2022, S. 31.
  - [12] Christoph Johann Stettina und Jeannette Hörz. „Agile portfolio management: An empirical perspective on the practice in use“. In: *International Journal of Project Management* 33.1 (2015), S. 151. ISSN: 0263-7863. DOI: <https://doi.org/10.1016/j.ijproman.2014.03.008>. URL: <https://www.sciencedirect.com/science/article/pii/S0263786314000489>.
  - [13] Dean Leffingwell u. a. „SAFe Reference Guide“. In: *Scaled Agile, Inc, [Online]*. Available: <https://www.scaledagileframework.com/>. [Accessed 1 June 2023] () .
  - [14] Klaus Leopold. *Agilität neu denken*. Deutschland: LEANability PRESS, 2019.
  - [15] Ralf Müller, Miia Martinsuo und Tomas Blomquist. „Project Portfolio Control and Portfolio Management Performance in Different Contexts“. In: *Project Management Journal* 39.3 (2008), S. 28–42. DOI: 10.1002/pmj.20053. eprint: <https://doi.org/10.1002/pmj.20053>. URL: <https://doi.org/10.1002/pmj.20053>.
  - [16] Husam Jasim Mohammed. „RETRACTED ARTICLE: The optimal project selection in portfolio management using fuzzy multi-criteria decision-making methodology“. In: *Journal of Sustainable Finance & Investment* 13.1 (2021), S. 125–141. DOI: 10.1080/20430795.2021.1886551. eprint: <https://doi.org/10.1080/20430795.2021.1886551>. URL: <https://doi.org/10.1080/20430795.2021.1886551>.

- [17] Pietari Pöntinen. „Recommended Guidelines for IT Project Portfolio Management Practices for The Case Companies“. In: (2019), S. 90–92. doi: <https://www.thesesus.fi/bitstream/handle/10024/172493/Thesis-P\b6ntinen-Pietari-IM-2019.pdf?sequence=2&isAllowed=y>.

## A Anhang

### A.1 Anhang 1

## Selbständigkeitserklärung

Hiermit erkläre ich, Maximilian Oedinger, dass ich die hier vorliegende Arbeit selbstständig und ohne unerlaubte Hilfsmittel angefertigt habe. Informationen, die anderen Werken oder Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich kenntlich gemacht und mit exakter Quellenangabe versehen. Sätze oder Satzteile, die wörtlich übernommen wurden, wurden als Zitate gekennzeichnet. Die hier vorliegende Arbeit wurde noch an keiner anderen Stelle zur Prüfung vorgelegt und weder ganz noch in Auszügen veröffentlicht. Bis zur Veröffentlichung der Ergebnisse durch den Prüfungsausschuss werde ich eine Kopie dieser Studienarbeit aufbewahren und wenn nötig zugänglich machen.