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

betreuender 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

Das ist mein Abstract.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Erklärung

Symbolverzeichnis

Symbol	Erklärung

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Kanban ist ein agiles Kommunikations-Framework, welches die Reaktionsfähigkeit und Effizienz eines Projektteams steigern soll. Dies wird durch einen Planungsprozess erreicht, der konstante neue Produktiterationen vorsieht und Arbeitsprozesse von Priorisierung und WIP-Limits abhängig macht. Klassisch wird Kanban in agilen Softwareentwicklungsprojekten mit Entwicklungsteams von 8 bis 12 Teammitgliedern angewendet. Die Flight-Level Methode beschränkt das Modell Kanban nicht mehr auf Projektteams, sondern sieht Anwendung in allen Unternehmensebenen, auch Flight-Level genannt, vor. Wird diese Methode erfolgreich auf allen Ebenen eingesetzt, erreicht die Organisation den sogenannten Status der Business-Agilität[1]. Die Methode wurde von Klaus Leopold entwickelt und beinhaltet diese drei Flight-Level, die im Weiteren betrachtet werden[2]:

1.2 Zielsetzung

1.3 Methodik

1.4 Gliederung der Arbeit

In dieser Arbeit werden zunächst die grundlegenden Prinzipien des agilen Gedanken erläutert und in einen Zusammenhang mit Projekt und Portfoliomanagement gebracht. Anschließenden werden Reporting und wertebasierte Entscheidungsfindung in verschiedenen Unternehmensbereichen erklärt und wie diese in einer gesamten Unternehmensstuktur Anwendung finden. Daraufhin wird ein Konzept erarbeitet, welches in einem Softwareprotoypen abgebildet wird, das die Dokumentation relevanter Informationen beliebig strukturierter Unternehmen erlaubt und somit automatisiertes Reporting wie z.B. Fortschrittsmessung ermöglicht. Ein Praxistest soll zuletzt den implementierten Protoypen evaluieren und eine kritische Reflexion auf das erarbeitete Konzept bieten.

2 Agile Unternehmensführung

2.1 Warum agil

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 Anforderungen vor der Umsetzung klar definiert und dokumentiert sind und somit Risiko minimiert wird. Diese Umstände sind allerdings nicht immer gegeben, bevor ein geplanter Prozess beginnen muss, damit Konkurrenzfähigkeit für ein Unternehmen gegeben ist. 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 erlaubt [3].

Bei traditioneller Planung erhöht sich durch diese Bedingungen das Risiko die falschen Dinge zum falschen Zeitpunkt zu tun. Agile Methodik erlaubt es diese Prozesse so effektiv wie möglich zu managen, da die Planung nicht linear, sondern iterativ stattfinde. Durch regelmäßige Feedbackschleifen mit Stakeholdern bleibt der Fokus auf Werteorientierung, da sich ändernde Anforderungen regelmäßig in den Planungsprozess der nächsten Iteration einbezogen werden. *Außerdem wird die Projektverantwortung von der Rolle des Projektmanagers ins Team gegeben.* Somit entsteht eine Flexibilität und Anpassbarkeit, welche die hohe Volatilität verringert. 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 [3].

Ziel bei der Wahl der Planungsmethode ist immer den Erfolg der Umsetzung des geplanten Prozesses zu maximieren. Für Projekte wird dieser Erfolg in zwei Schlüsselfaktoren unterteilt. Kurzfristiger Projekterfolg wird durch die Effizienz definiert, langfristiger Projekterfolg durch Effektivität. Diese beiden Faktoren werden durch Eingrenzung des Projektumfangs, schneller Lieferung, Qualitätssicherung, Kundenzufriedenheit und klare Kommunikation an und zwischen Stakeholdern [4]. Im Beispiel des Projektmanagements wurde bereits untersucht, inwiefern die Verwendung von agilen Methoden, den Projekt erfolgt steigert. Dabei stellte sich her-

aus, 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 [4]. Hat ein Projekt keinen Bedarf für agile Vorgehensweisen und wird dennoch agil durchgeführt, kann dies zu Verminderung des Projekterfolgs 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) []. Mit diesen funktionierenden und in sich geschlossenen Iterationen kann während der Umsetzung dann regelmäßig ein Feedbackprozess stattfinden. Diese Feedbackschleifen sind essenziell für die Erhaltung von Werteorientierung und Anpassbarkeit im Verlauf des Projekts [].

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. []

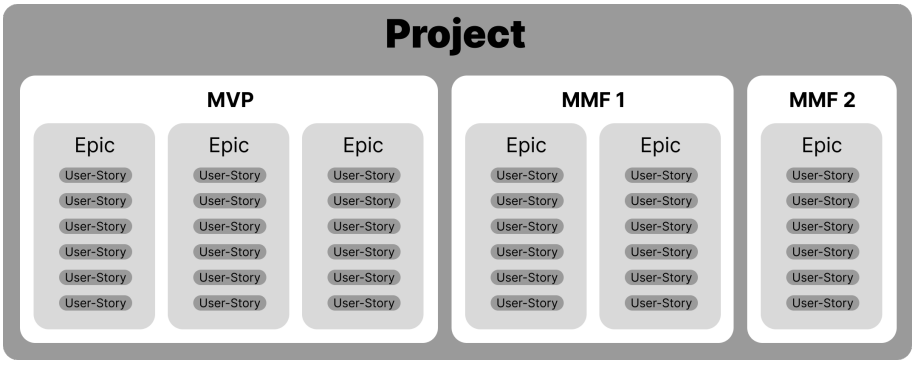


Abbildung 1: Agile Projektsegmentierung

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. []

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.

2.3 Agiles Portfoliomanagement

Der agile Ansatz kann ebenfalls für das Portfoliomanagement innerhalb eines Unternehmens verwendet werden. Traditionelles Portfoliomanagement oder auch Projekt Portfoliomanagement basiert auf ... [].

Agiles Portfoliomanagement dagegen hat das Ziel Unternehmensziele mit Initiativen oder Projekten zu verknüpfen und somit den Fluss von geleisteter Arbeit auf operativer Ebene zu steuern, um diese Ziele zu erreichen und dabei die Dynamik agiler Frameworks beizubehalten. []

2.4 Agile Unternehmensstrukturierung

Kombiniert man agile Projektstruktur und agiles Portfoliomanagement kann eine Organisation Business Agilität erreichen.

2.5 Beispiel Flight-Level

Flight-Level ist eine von Klaus Leopold entwickelte Methode, welche eine mögliche Form von agiler Unternehmensstrukturierung bietet. Die Methode Flight-Level wird hierbei von Leopold 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 Levels sollen einen Blick in verschiedenen 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 schnelle 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

3.1 Reporting in agilen Unternehmen

Reporting ist der Schlüsselfaktor für den Erfolg der Unternehmensorganisation durch z.B. agiles Portfoliomanagement, da es einen möglichst vollständigen und tiefgehenden Überblick erzeugen kann, auf Basis dessen Entscheidungen getroffen werden. [] Das Reporting sollte hierbei Einblick in verschiedene Detailgrade der Organisation bieten und somit für jeden Punkt, an dem Entscheidungen getroffen werden,...

3.2 qualitatives vs. quantitatives Reporting

Für effektives Reporting müssen verscheidene Metriken erhoben werden, hierbei unterscheided 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 automatisches Reporting

Reporting ist meist ein manueller Prozess, welcher mit immer wiederkehrendem Aufwand verbunden ist, da die Metriken regelmäßig erhoben werden müssen. Um das Reporting 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 konsistentere, regelmäßigere, validere und aktuellere Ergebnissen. Qualitative Metriken dagen 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 Konsitenz 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.

3.4 Reports in Portfoliomanagement

3.5 Reports für Value based Software-Engineering

3.6 Teamkoordination

4 Konzeption

Resultierend aus der Analyse soll an dieser Stelle ein Prozess definiert werden, aus denen anschließend ein UX-Konzept und 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 verwendbar 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. Aufgaben können außerdem 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 würden somit die Funktion der User-Stories erfüllen.

Die zuvor bereits beschriebene weitere Unterteilung in Tasks spielt für die Fortschrittsmessung oder Werteorientierung 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 können, muss es möglich sein, verschiedene Ebenen anzulegen, welche die hierarchische Struktur darstellen kann. 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.

4.2 UX-Entwurf für die Abbildung des Prozesses

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

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 hinterlegen zu können. 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. Werden Epics verwendet 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.

4.2.4 Dashboard

Das Dashboard zeigt alle Planungselemente einer Organisation hierarchisch angeordnet an 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.

4.2.5 Levels

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

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.

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 wird nicht nur ein Projekt-Element erstellt, sondern ebenfalls Aufgaben innerhalb des Projekts, die aus der Excel-Datei oder JIRA ausgelesen wurden. 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-Stati 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 Löschsymbol, mit dem das Element, nach einer Bestätigung, gelöscht werden kann. Durch einen Klick auf ein Element öffnet sich ein Pop-up-Pop-up-Fenster mit der Detailansicht der Aufgabe. Diese stellt den Aufgabennamen, die Beschreibung und Storypoints 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.

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 oder die Aufgabe löschen. Außerdem kann der Nutzer hier eine neue Aufgabe erstellen.

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.

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.

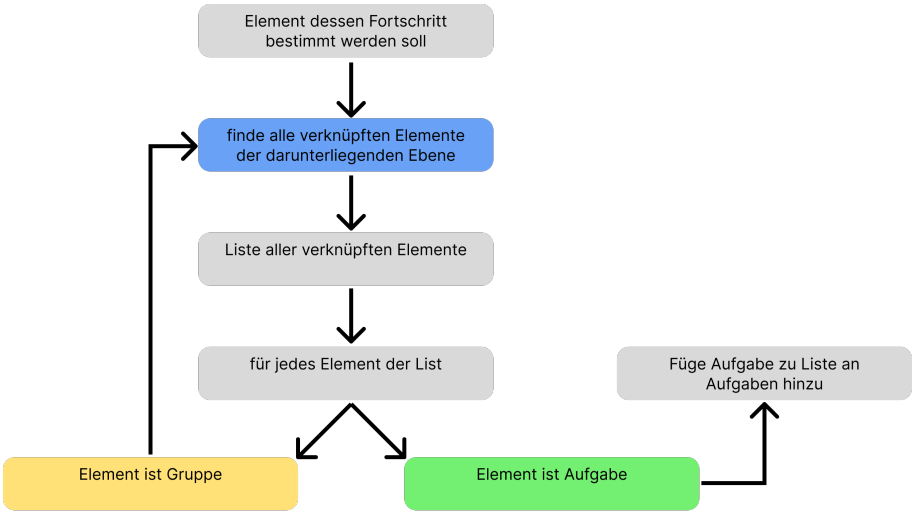


Abbildung 2: Prozess für Fortschrittsaggregation

Die Liste an Aufgaben, die sich am Ende des Prozesses ergibt, kann anschließend nach Aufgaben in den verschiedenen Stati sortiert werden, und entweder die Anzahl an fertigen Aufgaben durch die Gesamtmenge an Aufgaben geteilt, um denn generellen relativen Fortschritt zu bestimmen, oder die Summe der Storypoints aller fertigen Aufgaben durch die Gesamtmenge an Storypoints aller Aufgaben geteilt werden, um den absoluten Fortschritt zu bestimmen.

5 Implementierung

5.1 Datenstruktur

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

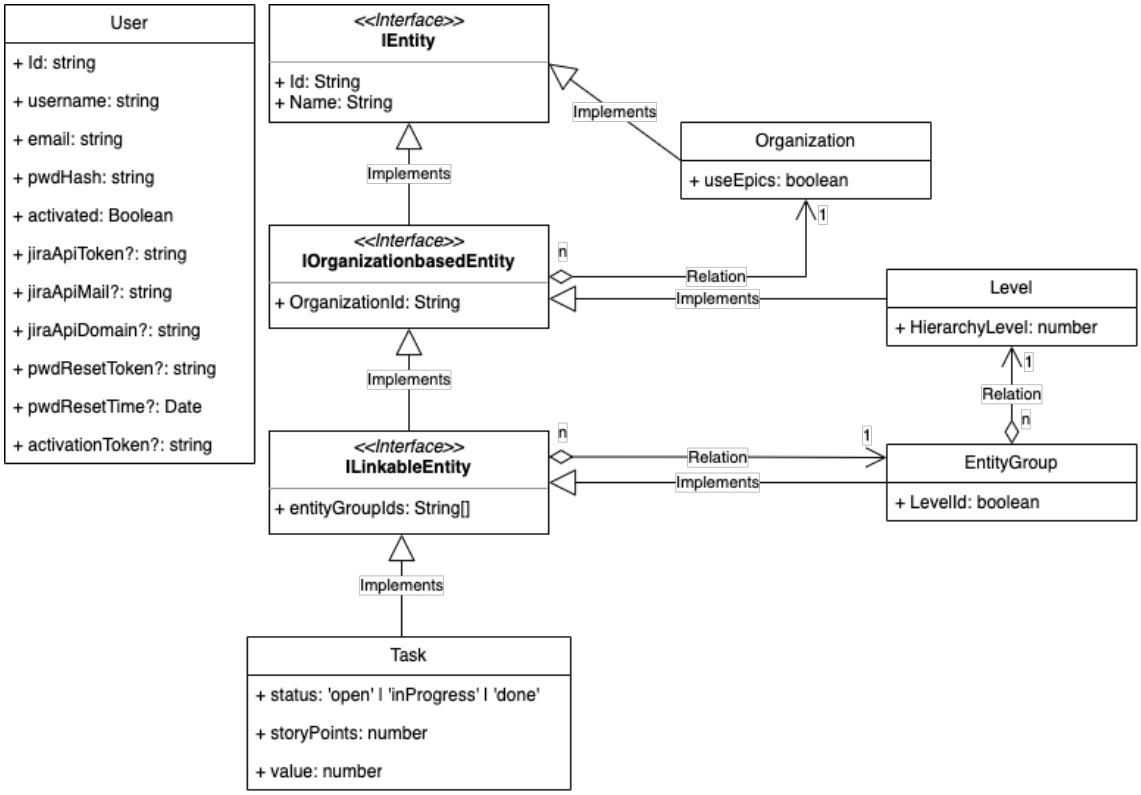


Abbildung 3: UML-Diagramm der Datenstruktur

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 und einen Namen 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 En-

titäten. 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.

5.2 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.

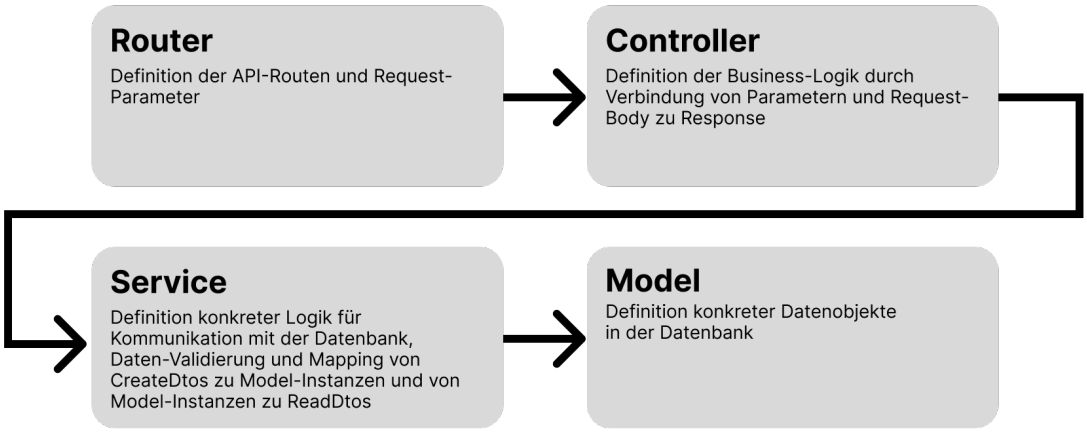


Abbildung 4: Backend Architektur

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.

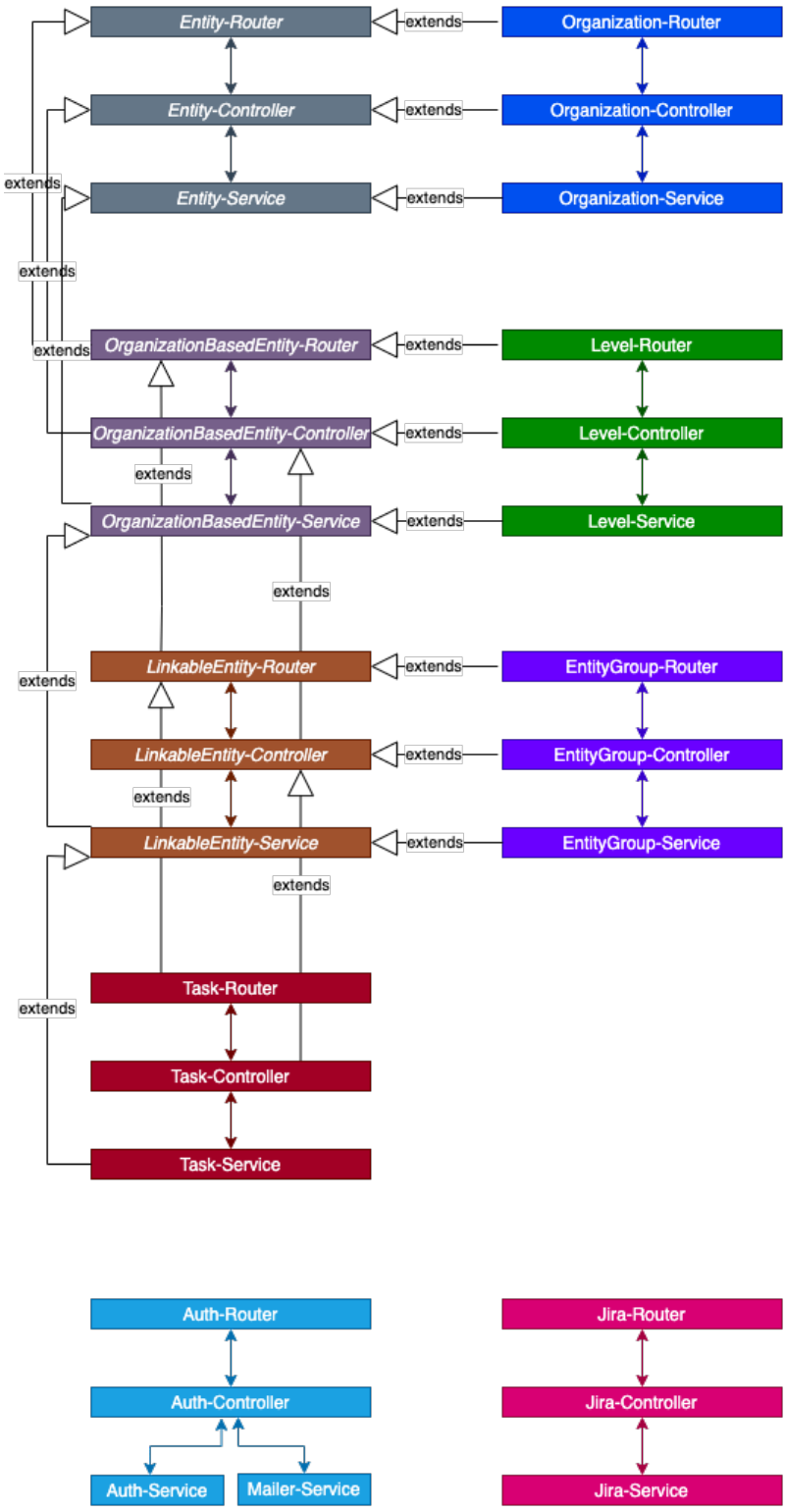


Abbildung 5: Abbildung der BE-Struktur

hier noch eine detaillierte Beschreibung der Strukturen ????

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.

5.3 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.

5.3.1 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. 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.

5.3.2 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.

Detailliertere Beschreibung der Components?????

5.3.3 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 simlple Stores wie z. B. den OrganizationStore beschränkt sich die absolute implementierung dann wie folgt:

Abbildung 6: Implementierung des OrganizationStore

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

```typescript
interface LevelGetters extends OrganizationBasedEntityGetters<ILevel> {
    getNextHierarchyLevel(): number;
    isProjectLevel(): (levelId: string) => boolean;
    getLowerLevel(): Level | undefined;
}

type LevelStore = OrganizationBasedEntityStore<
    ILevel,
    'level',
    OrganizationBasedEntityState<ILevel>,
    LevelGetters
>;
type LevelStoreDefinition = OrganizationBasedEntityStoreDefinition<
    ILevel,
    'level',
    OrganizationBasedEntityState<ILevel>,
    LevelGetters
>;

const levelStore: PiniaStore<LevelStore> = {
    state: makeOrganizationBasedEntityState<ILevel>(levelService),
    getters: {
        ...makeOrganizationBasedEntityGetters<ILevel>(),
        getNextHierarchyLevel(state) {
            const currentOrganization = useOrganizationStore().currentEntity;
            if (!currentOrganization) return 0;
            return state.entities.filter(
                ({ organizationId }) => organizationId === currentOrganization.id
            ).length;
        },
        isProjectLevel(state) {
            return (levelId: string) => {
                const projectLevel = useOrganizationStore().currentEntity?.useEpics
                    ? 1
                    : 0;
                return (
                    state.entities.find((level) => level.id === levelId)
                        ?.hierarchyLevel === projectLevel
                );
            };
        },
        getLowerLevel(state) {
            const currentLevel: Entity<OrganizationBasedEntity<ILevel>> =
                state.currentEntity;
            if (!currentLevel) return;

            return state.entities.find(
                (x) =>
                    x.organizationId === currentLevel.organizationId &&
                    x.hierarchyLevel === currentLevel.hierarchyLevel - 1
            );
        },
    },
    actions: makeOrganizationBasedEntityActions<ILevel>(),
};

export const useLevelStore: LevelStoreDefinition = defineStore(
    'level',
    levelStore
);
```

Abbildung 7: Implementierung des LevelStore

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.

5.4 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.

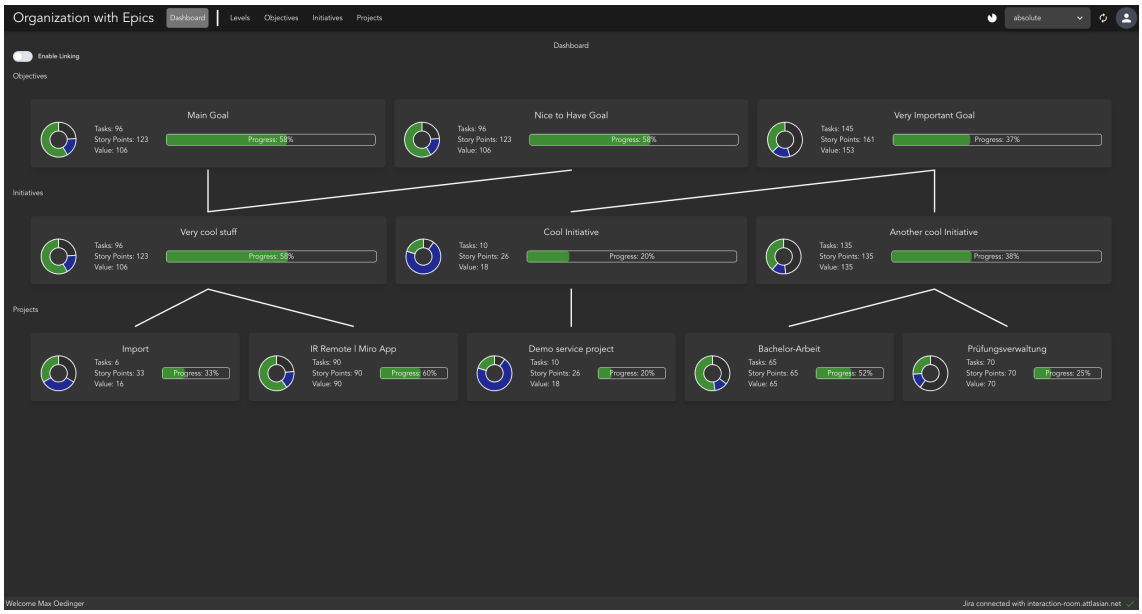


Abbildung 8: Dashboard

Um die Verbindungslinien darzustellen wurde ein SVG-Element im Hintergrund der Seite verwendet, welches die Verbindungslinien unabhängig von anderen Ele-

menten 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.

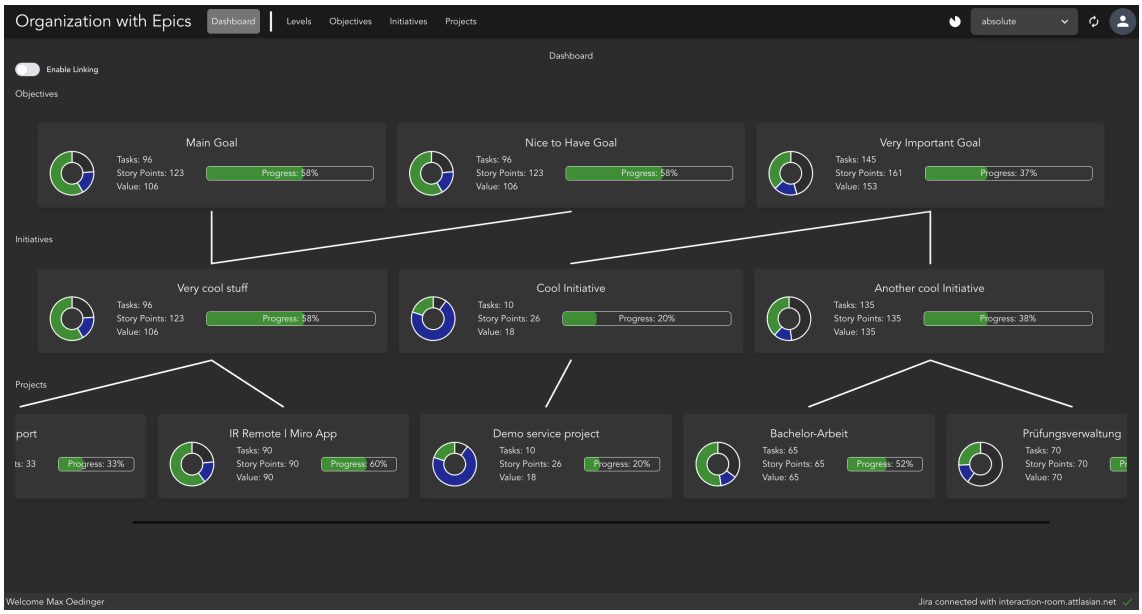


Abbildung 9: Dashboard mit scrollbaren Gruppen

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 Verbindungslinien 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 Verbindungslinien, blau hervorgehoben werden.

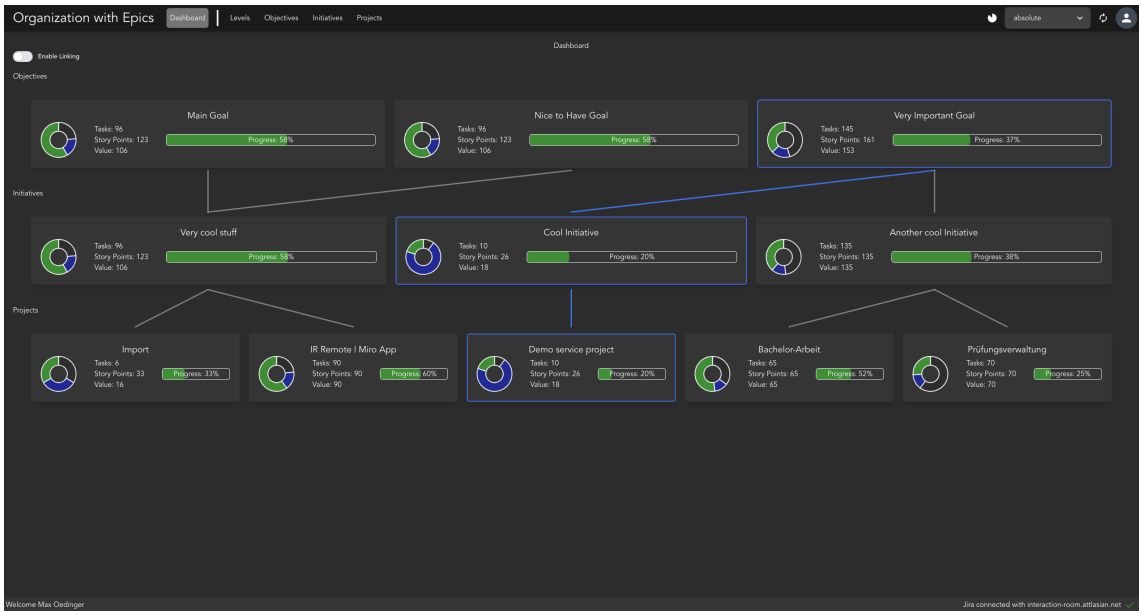


Abbildung 10: Dashboard mit hervorgehobenen Gruppen

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.

5.5 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.

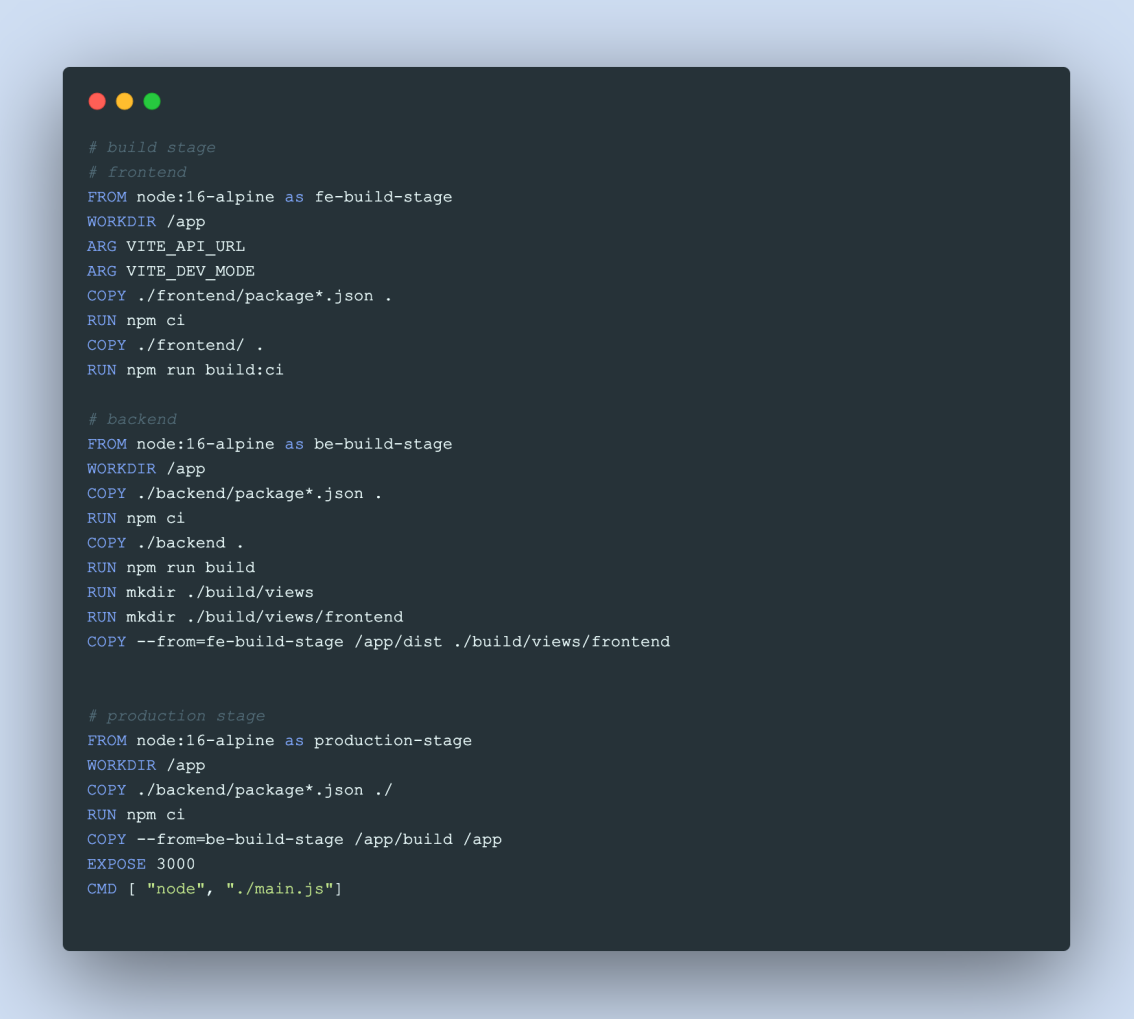
```yaml
name: Docker Image CI/CD Pipeline

on:
  pull_request:
    types:
      - closed
    branches:
      - main

jobs:
  build:
    if: github.event.pull_request.merged == true
    runs-on: self-hosted
    environment:
      name: production
    env:
      MONGO_PASSWORD: ${{ secrets.MONGO_PASSWORD }}
      SECRET: ${{ secrets.SECRET }}
      CLIENT_APP_URL: ${{ vars.CLIENT_APP_URL }}
      MONGO_DB: ${{ vars.MONGO_DB }}
      MONGO_URL: ${{ vars.MONGO_URL }}
      MONGO_USER: ${{ vars.MONGO_USER }}
      SMTP_HOST: ${{ vars.SMTP_HOST }}
      SMTP_PORT: ${{ vars.SMTP_PORT }}
      SMTP_USER: ${{ vars.SMTP_USER }}
      VITE_API_URL: ${{ vars.VITE_API_URL }}
      VITE_APP_DEV_MODE: ${{ vars.VITE_APP_DEV_MODE }}
    steps:
    - uses: actions/checkout@v3
    - name: Build the Docker image
      run:
        cd prototype &&
        docker build . --file "Dockerfile" --tag bachelor:latest
        --build-arg MONGO_PASSWORD=${{ env.MONGO_PASSWORD }}
        --build-arg SECRET=${{ env.SECRET }}
        --build-arg CLIENT_APP_URL=${{ env.CLIENT_APP_URL }}
        --build-arg MONGO_DB=${{ env.MONGO_DB }}
        --build-arg MONGO_URL=${{ env.MONGO_URL }}
        --build-arg MONGO_USER=${{ env.MONGO_USER }}
        --build-arg SMTP_HOST=${{ env.SMTP_HOST }}
        --build-arg SMTP_PORT=${{ env.SMTP_PORT }}
        --build-arg SMTP_USER=${{ env.SMTP_USER }}
        --build-arg SMTP_PASS=${{ env.SMTP_PASS }}
        --build-arg VITE_API_URL=${{ env.VITE_API_URL }}
        --build-arg VITE_DEV_MODE=${{ env.VITE_APP_DEV_MODE }}
  deploy:
    needs: build
    runs-on: self-hosted
    environment:
      name: production
    env:
      MONGO_PASSWORD: ${{ secrets.MONGO_PASSWORD }}
      SECRET: ${{ secrets.SECRET }}
      CLIENT_APP_URL: ${{ vars.CLIENT_APP_URL }}
      MONGO_DB: ${{ vars.MONGO_DB }}
      MONGO_URL: ${{ vars.MONGO_URL }}
      MONGO_USER: ${{ vars.MONGO_USER }}
      SMTP_HOST: ${{ vars.SMTP_HOST }}
      SMTP_PORT: ${{ vars.SMTP_PORT }}
      SMTP_USER: ${{ vars.SMTP_USER }}
      SMTP_PASS: ${{ secrets.SMTP_PASS }}
    steps:
      - name: Deploy the newest Docker image
        run: docker compose -f "prototype/docker-compose.ci.yml" up -d
      - name: Delete all old and unused Images
        run: docker system prune -af
```

Abbildung 11: 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.



```
# build stage
# frontend
FROM node:16-alpine as fe-build-stage
WORKDIR /app
ARG VITE_API_URL
ARG VITE_DEV_MODE
COPY ./frontend/package*.json .
RUN npm ci
COPY ./frontend/ .
RUN npm run build:ci

# backend
FROM node:16-alpine as be-build-stage
WORKDIR /app
COPY ./backend/package*.json .
RUN npm ci
COPY ./backend .
RUN npm run build
RUN mkdir ./build/views
RUN mkdir ./build/views/frontend
COPY --from=fe-build-stage /app/dist ./build/views/frontend


# production stage
FROM node:16-alpine as production-stage
WORKDIR /app
COPY ./backend/package*.json ./
RUN npm ci
COPY --from=be-build-stage /app/build /app
EXPOSE 3000
CMD [ "node", "./main.js"]
```

Abbildung 12: 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

7 Fazit

7.1 Ergebnis

7.2 Reflexion

7.3 Ausblick

Literatur

[1] Klaus Leopold. *Agilität neu denken*. Deutschland: LEANability PRESS, 2019.

[2] Ursula Kusay-Merkle. *Agiles Projektmanagement im Berufsalltag - Für mittlere und kleine Projekte*. Deutschland: © Springer-Verlag GmbH, ein Teil von Springer Nature, 2018, S. 210.

[3] 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.

[4] 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.

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.