Technische Dokumentation des Projekts „Taskitory“

Programmentwurf für die Vorlesung Software Engineering II

für die Prüfung zum

Bachelor of Science

des Studiengangs Informatik

Studienrichtung Angewandte Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Fabian Schwickert

16. Mai 2022

Matrikelnummer 4439027

Kurs TINF19 B4

Ausbildungsfirma Siemens AG, Karlsruhe

Gutachter der Studienakademie Mirko Dostmann

Inhaltsverzeichnis

[Inhaltsverzeichnis II](#_Toc103617409)

[Abbildungsverzeichnis IV](#_Toc103617410)

[1 Problemstellung, Ziel und Aufbau 1](#_Toc103617411)

[2 Clean Architecture 2](#_Toc103617412)

[2.1 Grundlagen 2](#_Toc103617413)

[2.2 Implementierung in der Applikation „Taskitory“ 3](#_Toc103617414)

[3 Domain Driven Design 6](#_Toc103617415)

[3.1 Problemdomäne 6](#_Toc103617416)

[3.2 Ubiquitous Language 8](#_Toc103617417)

[3.2.1 Begriffe der Fachlichkeit 8](#_Toc103617418)

[3.2.2 Begriffe der technischen Implementierung 13](#_Toc103617419)

[3.3 Taktische Muster des Domain Driven Design 16](#_Toc103617420)

[3.3.1 Entitäten 17](#_Toc103617421)

[3.3.2 Value Objects 22](#_Toc103617422)

[3.3.3 Aggregate 25](#_Toc103617423)

[3.3.4 Repositories 27](#_Toc103617424)

[3.3.5 Domänen Services 27](#_Toc103617425)

[4 Use Cases 27](#_Toc103617426)

[4.1 Projekte 27](#_Toc103617427)

[4.2 Projekt-Mitgliedschaften 28](#_Toc103617428)

[4.3 Kanban Boards 29](#_Toc103617429)

[4.4 Nachrichten 30](#_Toc103617430)

[4.5 Aufgaben 30](#_Toc103617431)

[4.6 Benutzer 31](#_Toc103617432)

[4.7 Tags 32](#_Toc103617433)

[5 Entwurfsmuster 32](#_Toc103617434)

[6 Programmierprinzipien 35](#_Toc103617435)

[6.1 SOLID 36](#_Toc103617436)

[6.1.1 Single Responsibility Principle 36](#_Toc103617437)

[6.1.2 Open Closed Principle 36](#_Toc103617438)

[6.1.3 Liskov Substitution Principle 36](#_Toc103617439)

[6.1.4 Interface Segregation Principle 37](#_Toc103617440)

[6.1.5 Dependency Inversion Principle 37](#_Toc103617441)

[6.2 GRASP 38](#_Toc103617442)

[6.2.1 Geringe Kopplung 38](#_Toc103617443)

[6.2.2 Hohe Kohäsion 38](#_Toc103617444)

[6.2.3 Information Expert 38](#_Toc103617445)

[6.2.4 Polymorphie 39](#_Toc103617446)

[6.2.5 Pure Fabrication 39](#_Toc103617447)

[6.2.6 Delegieren 39](#_Toc103617448)

[6.2.7 Controller 40](#_Toc103617449)

[6.2.8 Creator Principle 40](#_Toc103617450)

[6.3 DRY 40](#_Toc103617451)

[7 Code Smells und Refactoring 41](#_Toc103617452)

[7.1 Open Closed Principle Verstoß 41](#_Toc103617453)

[7.2 DRY Prizip Verstoß 45](#_Toc103617454)

Abbildungsverzeichnis

Abb. 1: Die Problemdomäne der Applikation „Taskitory“ 8

Abb. 2: Klassen „Task“ und „KanbanBoard“ ohne Erbauer Muster 34

Abb. 3: Die Kanbanboard Entität mit integriertem Erbauer Muster 35

Abb. 4: Die Nachrichten Entität 42

Abb. 5: Die konkreten Nachrichten Implementierungen 43

Abb. 6: Die Factory-Methoden für das erzeugen konkreter Nachrichten 44

Abb. 7: Die Konstruktoren des Value Objects „Sprint“ (Teil 1) 45

Abb. 8: Die Konstruktoren des Value Objects „Sprint“ (Teil 2) 46

Abb. 9: Die refactorierten Sprint-Konstruktoren (Teil 1) 46

Abb. 10: Die refactorierten Sprint-Konstruktoren (Teil 2) 47

Abb. 11: Die extrahierte Validierungslogik für Sprints 47

# Problemstellung, Ziel und Aufbau

Es gibt eine Reihe von Aufgabenverwaltungs-Systemen wie z. B. Jira, Youtrack oder Mi­cro­soft Planner, die mit einem Kanban Board arbeiten. Die hier dokumentierte Klau­sur­er­­­satz­leis­tung besteht in der Entwicklung einer App­li­ka­tion, die eine solche Auf­ga­ben­ver­wal­­­tung mo­delliert. Es wird kei­ne graphische Benutzeroberfläche (GUI) entwickelt und le­­diglich die not­wendige An­wen­dungs­logik pro­grammiert und über einen REST-Service (Re­­pre­sentational State Transfer) verfügbar ge­macht. Für die Implementierung des Rest-Ser­vices wird die Pro­gram­miersprache Java mit dem Package Manager „Ma­­ven“ und dem „Spring-Boot-Framework“ verwendet.

Mit der Anwendung soll es möglich sein, die Aufgaben in einem Projekt gemeinsam in ei­nem Projekt-Team zu verwalten. Dazu müssen Benutzer Accounts anlegen, die in den Kon­­­text gemeinsamer Projekte gebracht werden. Das Identity- und Accessmanagement (IAM) soll dabei durch einen Keycloak umgesetzt werden. Ei­ne persistente Da­ten­spei­che­rung wird mit einer Postgresql-Datenbank (DB) um­ge­setzt. In der DB werden die Daten des Keycloaks und ein eigenes DB-Modell gehalten. Die benannten Komponenten (Key­cloak, DB und REST-Service) der App­likation sollen do­cke­risiert werden und mit „Docker-Com­pose“ konfiguriert und be­trie­ben werden.

Um die Bedienung des REST-Services zu vereinfachen, wird eine Sammlung von Anfragen und Konfigurationen für einen HTTP-Client (Hypertext Transfer Protocol) zur Verfügung gestellt. Dazu wird eine „Collection“ für Anwendung Postman zur Verfügung gestellt.

Alle Inhalte der hier dokumentierten Klausurersatzleistung sind auf GitHub unter dem Link „https://github.com/Krayaty/Taskitory“ erreichbar.

# Clean Architecture

## Grundlagen

Unter „Clean Architecture“ versteht man einen Architektur-Ansatz für Software-Applikationen. Die Clean Architecture fußt auf der Idee, dass Applikationen in mehrere funktional abgegrenzten Schichten aufgeteilt werden können. Wie bei einer Zwiebel um­schließt eine Schicht die andere und nur die äußerste Schicht ist für Benutzer der App­li­ka­tion zugänglich. Die Funktionen der inneren Schichten sind über definierte Schnitt­stel­len für die jeweils folgende Schicht zugänglich. Die äußeren Schichten sind dadurch von den inneren Schichten abhängig; umgekehrt sind die inneren jedoch nicht von den äu­ße­ren Schichten abhängig. Daher wird es schwerer den Code auf einer Schicht zu ändern, je wei­ter innen sie liegt. Die Clean Architecture ermöglicht durch den be­schrie­be­nen mo­du­la­ren Aufbau, dass die für eine Domäne und Applikation spezifische Ge­schäfts­logik nicht ge­ändert werden muss, wenn konkrete technische Implementierungen ei­ner Applikation wie geändert werden müssen. Auf diese Weise kann ein Applikation lang­fristig mit wenig Auf­wand gewartet und angepasst werden.

Im Allgemeinen werden die folgenden fünf Schichten der Clean Architecture definiert (von innen nach außen):

1. Abstraktions-Schicht:

Die Abstraction-Schicht ist der Kern einer Applikation. In dieser Schicht wird sog. do­mä­nen­­übergreifendes Wissen implementiert. Das meint Algorithmen und Konzepte, die nicht an die Domäne einer Applikation gebunden sind. Zudem werden hier Software-Werk­­zeuge und -Bibliotheken ein­gebunden, die auf allen folgenden Schichten verwendet wer­­den und technologische Standards umsetzen.

1. Domänen-Schicht:

Die Domänen-Schicht umschließt die Abstraktions-Schicht und hängt daher von ihr ab. In der Domänen-Schicht wird die Fachlichkeit einer Applikation umgesetzt. Hier wird die übergreifende Geschäftslogik implementiert, die für die Abbildung einer Domäne in vielen verschiedenen App­li­kation benötigt wird. Die Domänen-Schicht wird durch das Domain Driven Desing (siehe Ka­pitel 3) modelliert.

1. Applikations-Schicht:

Die Applikations-Schicht folgt direkt auf die Domänen-Schicht und hängt daher von ihr ab. In der Applikations-Schicht werden die Use Cases einer Applikation abgebildet. In diesem Teil einer Applikation werden die für den Benutzer erfahrbaren Funktionen im­ple­mentiert. Das umfasst auch die Implementierung der Applikations-spezifischen Ge­schäfts­logik einer Domäne.

1. Adapter-Schicht:

Die Adapter-Schicht baut auf der Applikations-Schicht auf und hängt daher von ihr ab. Der Zweck der Adapter-Schicht ist es, Benutzerinteraktionen an die Applikations-Schicht wei­ter­zugeben. Dabei dient die Adapter-Schicht als „Anti Corruption Layer“, der Domänen- und Applikations-spezifischen Quellcode von festen technischen Implementierungen ab­kop­pelt. Das meint auch die Konvertierung der Daten aus eingehenden Anfragen in ein für die darunter liegenden Schichten verständliches Format.

1. Plugin-Schicht:

Die Plugin-Schicht liegt ganz außen und hängt von der darunter liegenden Adapter-Schicht ab. Auf der Plugin-Schicht werden technische Implementierungen wie z. B. das Ein­binden einer Datenbank oder eines IAM-Services vorgenommen. Außerdem werden auf dieser Schicht die Funktionen einer Applikation den Benutzern zugänglich gemacht.

## Implementierung in der Applikation „Taskitory“

Die Applikation Taskitory verwendet das Konzept der Clean Architecture mit allen oben be­schriebenen Schichten. Um die Eigenschaft der nach innen gerichteten Abhängigkeiten in der Applikation Taskitory sicherzustellen, wird für jede Schicht ein eigenes Maven- Mo­dul angelegt. Die Maven-Module werden wie oben beschrieben über Ab­hän­gig­kei­ten miteinander verbunden. Nachfolgend wird auf alle Schichten und deren Aufgabenfelder ein­gegangen.

1. Abstraktions-Schicht:

Die Abstraction-Schicht der Applikation Taskitory enthält keinen eigenen Quellcode. Lediglich die folgenden Abhängigkeiten werden auf dieser Schicht eingebunden:

* Java Persistence API (JPA)
* Apache Commons-Lang
* Org.JSON
* Spring-Context
* Lombok

Die „Java Persistence API“ (JPA) ist ein de fakto Standard für die Integration einer Da­ten­bank in eine Java-Applikation. Da die JPA eine weit verbreitete, beständige Software-Bib­lio­thek ist, ist es nicht notwendig sie als Teil der Plugin-Schicht einzubinden. Denn das wür­­de umfassende Implementierungen auf der Adapter-Schicht nach sich ziehen. Die JPA hat keinen Bezug zu der Domäne der Applikation Taskitory und kann als Standard-Tech­nik angesehen werden. Daher gehört diese Abhängigkeit auf die Abstraktions-Schicht.

Die Software-Bibliotheken Apache Commons-Lang und Org.JSON stellen Funktionalitäten für die Ver­ar­bei­tung von Inhalten zur Verfügung, die nicht von den Java Standard-Bib­lio­the­ken mitgeliefert werden. Mit Apache Commons-Lang werden all­­gemeine Pro­blem­stel­lun­gen wie z. B. das Hashen erledigt. Org.JSON auf die Verarbeitung von Inhalten im JSON-For­mat. Diese allgemeinen Funk­tio­nen können für alle folgenden Schichten re­le­vant sein und sind nicht an eine bestimmte Domäne gebunden. Daher werden diese Abhängigkeit auf der Abstraktions-Schicht ein­ge­bun­den.

Die Abhängigkeiten „Spring-Context“ und „Lombok“ sind zwei Software-Bibliotheken, de­ren Funktionen allen anderen Schichten zur Verfügung stehen soll. Mit Spring-Context wer­­den Funktionen zur Verfügung gestellt, die das Betreiben einer eigenen Applikation mit dem Spring-Framework ermöglichen. Lombok ist eine Software-Bibliothek mit der Quellcode vereinfacht werden kann. Lombok ermöglicht es, Stan­dard-Konstrukte in Java wie z. B. Getter und Setter durch eine Annotation zu ersetzen. Die Funk­tionen der beiden Software-Bibliotheken Spring-Context und Lombok beziehen sich auf allgemeine Pro­gram­mier-Konzepte und Konstrukte der Programmiersprache Java. Da­her gehören beide Abhängigkeiten auf die Abstraktions-Schicht.

1. Domänen-Schicht:

Die Domänen-Schicht ist das Herzstück der Applikation Taskitory. Hier wird die grund­le­gen­­de Geschäftslogik der Domäne definiert. Es wird ein statisches Modell auf­gestellt, das die fachlichen Akteure der Applikation Taskitory definiert. Das umfasst besonders die En­ti­­täten „Benutzer“, „Projekt“, „Projekt-Mitgliedschaft“, „Auf­ga­be“, „Kanbanboard“, „Tag“, „Nach­richt“. In Kapitel 3 werden diese Entitäten näher beschrieben. Die Applikation Tas­ki­tory kommuniziert über JPA mit einer Datenbank und persistiert so die anfallenden re­le­vanten Daten.

Der Einsatz von JPA erfordert ein „Objektrelationales Mapping“ (OR-Mapping). Dabei wer­den die Tabellen einer Datenbank durch Klassen und die Datensätze durch Instanzen die­ser Klassen repräsentiert. In der Applikation Taskitory müssen die auf der Do­mä­nen­schicht definierten Entitäten (siehe oben) persistiert werden. Um für diese Entitäten ein OR-Mapping herstellen zu können, werden technische Implementierungen von JPA in den Do­mänen-Quellcode eingebracht. Es wurde bewusst entschieden das OR-Mapping auf der Do­mänen-Schicht umzusetzen, da eine Implementierung auf der Plugin-Schicht zu den Do­mänen-Klassen äquivalente Klassen erfordert hätte. Diese räumliche Trennung von Fach­lichkeit und technischen Implementierungen wird nicht durch den deutlich höheren Im­plementierungsaufwand gerechtfertigt.

Es werden JPA-Annotationen verwendet, die dem fachlichen Quellcode der Domäne an­ge­heftet werden. Dadurch kann der Domänen-Quellcode auch weiterhin von technischen Im­plementierungen getrennt betrachtet werden.

1. Applikations-Schicht:

Auf Grundlage des statischen Modells der Akteure aus der Domänen-Schicht wird in der App­likations-Schicht ein dynamisches Modell der Abläufe aufgestellt. Hier werden die Use Ca­ses der Applikation Taskitory umgesetzt. Dabei wird der Informationsfluss von Akteur zu Akteur unter Be­rück­sich­tigung der Geschäftslogik aus der Domänen-Schicht ab­ge­bil­det. Die Use Cases werden zudem durch Applikations-spezifische Geschäftslogik gelenkt. Die wichtigsten Use Cases der Applikation Taskitory sind in Kapitel 4 beschrieben.

1. Adapter-Schicht:

Die Applikation Taskitory benutzt für Benutzerinteraktionen eine REST-Schnittstelle. Es gibt kein Frontend, das die Validierung von Anfrage-Parametern übernimmt. Da­her müs­sen die eingehenden HTTP-Anfragen in dem Backend validiert und in passende Formate um­­ge­wandelt werden, bevor sie an die inneren Schichten der Applikation weiter­ge­ge­ben wer­den. Für diesen Zweck ist die Adapter-Schicht gedacht. Hier befindet sich der Quellcode, mit dem die HTTP-Anfragen der REST-API in Methodenaufrufe und Da­ten­trans­fer­ob­jekte (DTO) der Applikationsschicht umgewandelt werden. Dieser Umwandlungs-Prozess findet unter Beachtung der Validierungs-Logik statt.

1. Plugin-Schicht:

Die Plugin-Schicht der Applikation Taskitory enthält die technischen Implementierungen für die Kommunikation mit einer Postgresql Datenbank und einem Keycloak IAM-Server. Da­zu wird das Persistenz-Framework „Hibernate“ und das Sicherheits-Framework „Spring-Security“ verwendet. Zu­dem werden die Funktionen der Applikation Taskitory auf dieser Schicht den Benutzern über eine REST-Schnittstelle ver­fügbar gemacht.

# Domain Driven Design

## Problemdomäne

Die Problemdomäne einer Applikation setzt sich aus einer Kerndomäne, mehreren un­ter­stüt­zenden und mehreren ge­nerischen Domänen zusammen. Die Kerndomäne bestimmt die zentralen Funk­tio­na­li­tä­ten und den Zweck einer Applikation. Unterstützenden Do­mä­nen beziehen sich auf Aufgaben-Felder, die nicht direkt Teil der Kerndomäne sind, aber die Kerndomäne mit ihren Funktionen unterstützen. Die Kerndomäne verwendet die un­ter­stützenden Domänen, um die zentralen Funktionen einer Applikation umzusetzen. Die Ge­nerischen Domänen beziehen sich auf Aufgabenfelder, die nicht direkt mit dem Zweck ei­ner Applikation in Verbindung stehen, jedoch für das Betreiben einer Applikation not­wen­dig sind. Die Generischen Domänen liefern Basis-Funktionalitäten, ohne die eine App­li­kation nicht lauffähig ist.

Die Problemdomäne der Applikation „Taskitory“ ist die Aufgaben­ver­waltung in Projekten mit mehreren Projekt-Teilnehmern. Dabei ist die Kerndomäne das Verwalten von Auf­ga­ben. Das Umfasst nicht nur das Erstellen, Verändern und Löschen von Aufgaben, sondern auch die Funktionen eines Kanban Boards.

Die über die Kerndomäne zur Verfügung gestellten Daten und Funktionen werden Be­nut­zern der Applikation über die Domäne IAM zur Verfügung gestellt. Ohne eine Be­nut­zer­ver­waltung könnte das Konzept von Projekt-Teams und damit die ganze Applikation nicht um­gesetzt werden. IAM hat keine inhaltliche Verbindung zu der Kerndomäne und wird daher als generische Domäne eingestuft.

Die Verwaltung von Pro­jekten ist kein zwingend notwendiger Aufgabenbereich für eine Auf­­­gabenverwalten. Allerdings ist dieser Aufgabenbereich bei der Applikation „Tas­ki­tory“ relevant für die Funktionen der Kerndomäne und inhaltlich mit dieser verbunden. Denn alle Objekte, die in der Kerndomäne verwendet, erzeugt oder gelöscht werden, sind ei­­­nem Projekt zugeordnet. Auch die Rolle eines Benutzers in einem Projekt kann in diese Do­­­mäne gezählt werden. Denn der Zugriff auf die Funktionen der Kerndomäne wird durch die Rolle eines Be­­nutzers in einem Projekt eingeschränkt. Auf Grund der in­halt­li­chen Verbindung zur Kerndomäne wird die Verwaltung von Projekten und der damit as­so­ziierten Projekt-Teams als un­ter­stüt­zen­­de Domäne eingestuft.

Das Erstellen von Statistiken ist keine für eine Aufgabenverwaltung notwendige Funktion. Al­­lerdings werden mit Statistiken die Daten von Kanban Boards ver­ar­bei­tet, die aus der Kern­domäne stammen. Es besteht damit eine klare inhaltliche Verbindung zur Kern­do­mä­ne. Das Erstellen von Statistiken ist damit eine unterstützende Domäne.

Die Applikation „Taskitory“ informiert Benutzer mit Nachrichten über bestimmte Er­eig­nis­­se, die für den Benutzer relevant sind. Diese Ereignisse betreffen i. d. R. nicht die Kern­do­­mäne, sondern unterstützende Domänen. Es besteht daher keine inhaltliche Ver­bin­dung zur Kerndomäne. Das Messaging ist eher eine Basisfunktionalität und wird daher als ge­nerische Domäne eingestuft. Abbildung 1 zeigt noch einmal die Problemdomäne der App­likation „Taskitory“ mit allen Bestandteilen.

**Problemdomäne**

IAM

**Generische Domänen**

**Unterstützende Domänen**

Projekt-Verwaltung

**Kerndomäne**

Aufgabenverwaltung

Statistiken

Messaging

Abb. 1: Die Problemdomäne der Applikation „Taskitory“

## Ubiquitous Language

Die „Ubiquitous Language“ ist eine Sammlung von Begriffen, mit denen eine Applikation wird. Das um­fasst Fachbegriffe der Pro­blem­domäne und der technischen Im­ple­men­tie­rung. Es müssen nicht triviale Objekte, Kon­zepte und Regeln eindeutig definiert werden.

### Begriffe der Fachlichkeit

Die Ubiquitous Language der Applikation Taskitory umfasst die folgenden Be­griffe für das Be­­schreiben problemdomänenspezifischer Objekte, Konzepte und Regeln.

1. Projekt

Ein Projekt ist ein zielgerichtetes Vorhaben, das aus einer Menge von Aufgaben besteht, die von einer Gruppe von Benutzern verwaltet und bearbeitet werden. Ein Projekt hat eine eindeutige Bezeichnung und eine Beschreibung, die von Pro­jekt-Ad­mi­­nistratoren festgelegt und verändert werden kann. Für die Erfüllung die­ser Aufgaben, stellen Projekte Kanbanboards und Tags zur Verfügung. Projekte sind der zen­trale Kontext, in dem ein Benutzer die Applikation Taskitory verwendet.

Über die Zeit können Benutzer ein Projekt verlassen und wieder beitreten. Für das Ver­las­­sen eines Projekts wird keinerlei Zustimmung von Seiten des Projekt-Teams benötigt. Al­­lerdings kann ein Benutzer einem Projekt nur dann beitreten, wenn er von einem Projekt-Administrator eingeladen wird.

1. Projekt-Einladung

Eine Einladung in ein Projekt kann nur von einem Projekt-Administrator ausgestellt wer­den. Eine Einladung ist eine Instanz der Nachrichten-Entität und ist daher genau einem Pro­­jekt und einem Benutzer zugeordnet. Eine Einladungs-Nachricht enthält den ge­hei­men Projekt-Schlüssel in gehashter Form und die Rolle, die der Benutzer im Pro­jekt ein­neh­men soll.

1. Projekt-Schlüssel

Der Schlüssel eines Projekts ist eine geheime Zeichenkette, die nur von Projekt-Ad­mi­nis­tra­toren verwaltet werden kann. Der Schlüssel eines Projekts ist nur gehasht einsehbar und kann ausschließlich auf einen zufällig generierten Wert geändert werden.

1. Benutzer

Ein Benutzer wird durch ein Benutzer-Konto repräsentiert. Dieses Benutzer-Konto be­steht aus einem Benutzernamen und einem Passwort. Beide Werte können durch den Be­nutzer selbst verändert werden.

Be­nutzer müssen keinem Projekt angehören, können aber beliebig vielen Projekten bei­tre­­­ten. Benutzer können die Applikation Taskitory ohne eine Projektmitgliedschaft nicht sinn­­voll nutzen. Daher können Benutzer jederzeit neue Projekte erstellen, in denen sie au­tomatisch als Projekt-Administrator eingetragen werden.

Es werden der Beginn von Projektmitgliedschaften und die eingenommene Rolle in einem Pro­­jekt festgehalten. Benutzer können die Rollen „Mitglied“ und „Admin“ einnehmen.

1. Projekt-Administrator

Pro­jekt-Administratoren sind Benutzer mit der Admin-Rolle. Diese Rolle ist pro­jekt­spe­zi­fisch. Benutzer mit der Admin-Rolle haben alle Rechte eines Mitglieds und zu­sätzliche Rechte für die Projekt-Verwaltung.

Es muss zu jeder Zeit einen Ad­mi­nis­tra­tor in einem Projekt ge­ben. Wenn der letz­te Ad­mi­nis­­trator ein Projekt verlässt, muss er einen neuen Projekt-Administrator aus dem Pro­jekt-Team bestimmen. Wenn es keine anderen Team-Mitglieder gibt, wird das Pro­jekt ge­löscht. Es gibt keine Ober­gren­­ze für die Anzahl von Projekt-Administratoren. Nur Projekt-Ad­­­­ministratoren können Benutzer ein­la­den, entfernen und deren Rolle än­dern.

1. Projekt-Mitglieder

Projekt-Mitglieder sind Benutzer mit der Mitglieds-Rolle. Diese Rolle ist pro­jekt­spe­zifisch. Die Projekt-Mitgliedschaft ermächtigt einen Benutzer die Aufgaben und Kanban­boards eines Pro­­jekts zu verwalten.

1. Aufgabe

Auf­gaben sind Teil eines Projekts und beschreiben eine Tätigkeit, die durchgeführt wer­den soll. Bei Erfüllung aller Aufgaben wird ein über­ge­­ordnetes Pro­jekt-Ziel erreicht.

Auf­gaben sind einem Projekt zugeordnet und haben eine im Projekt eindeutige Be­zeich­nung und eine nicht eindeutige Beschreibung. Über einen Erstellungs- und einen Fer­tig­stel­lungs-Zeit­punkt wird der Bearbeitungszeitraum einer Aufgabe festgehalten. Der Be­ar­beitungs-Sta­tus einer Aufgabe ist eng damit verbunden. Der Bearbeitungs-Status kann die Werte der Spalten eines Kanbanboards (siehe unten) annehmen. Die Kom­­ple­xi­tätund die Priorität in der Bearbeitungsreihenfolge werden jeweils über einen eigenen Wert fest­ge­halten. Ei­ne Aufgabe enthält Informationen über den Autor der Aufgabe und die für die Be­­ar­bei­tung aktuell zuständige Person. Die Aufgaben werden neben diesen At­­­tri­­bu­ten auch durch Tags mit Meta-Informationen beschrieben.

Aufgaben sind immer einem Projekt zugeordnet. Aufgaben können sich entweder auf ei­nem oder keinem Kanbanboard befinden. Wenn eine Aufgabe nicht auf einem Kann­ban­board zu finden ist, liegt sie im Backlog. Aufgaben können entweder direkt auf einem Kan­ban­­board erzeugt werden oder vorerst im Backlog abgelegt werden.

1. Backlog

Das Backlog eines Projekts ist die Sammlung aller Aufgaben, die keinem Kanbanboard zu­ge­ordnet sind. Aus dem Backlog können Aufgaben einem beliebigen Kanbanboard des glei­chen Projekts zugewiesen werden. Aufgaben, die von einem Kanbanboard entfernt wer­den, werden automatisch in das Backlog zurückgelegt.

1. Kanban Board

Eine Kanbanboard ist eine Tabelle, bei der jede Spalte einen Bearbeitungs-Status re­prä­sen­­tiert. Aufgaben können in diese Spalten einsortiert werden, um den Bearbeitungs-Sta­tus der Aufgabe zu visualisieren. Um einen Fortschritt ab­zu­bil­den, werden die Aufgaben von einer in eine andere Spal­­te geschoben.

Kanbanboards sind einem Projekt zugeordnet und haben eine im Projekt eindeutige Be­zeich­nung und eine nicht eindeutige Beschreibung. Ein Kanban Board be­trifft einen be­stimm­­ten „Sprint“ genannten Zeitraum. Dieser Zeit­raum wird durch einen Start- und End-Zeit­punkt festgehalten. Nach einem Sprint besteht das Kan­ban Board weiter, kann aber nicht mehr bearbeitet werden. Ein al­tes Kan­ban Board kann in ein neues Kanban Board über­führt werden, das bearbeitet werden kann.

Kanbanboards können grundsätzlich eine beliebige Anzahl Spalten haben. Weit verbreitet sind die folgenden fünf Spalten (Die Spalten werden in der unten genutzten Reihenfolge auf einem Kanbanboard angeordnet).

* Todo

In dieser Spalte werden alle Aufgaben abgelegt, die sich nicht aktuell in Bearbeitung be­finden. Aufgaben haben bei ihrer Erzeugung diesen Status. In der App­­likation Tas­ki­to­ry ist diese Spalte immer sichtbar.

* In Progress

In dieser Spalte werden alle Aufgaben abgelegt, die aktuell bearbeitet werden. In der App­­likation Taskitory ist diese Spalte immer auf einem Kanbanboard sichtbar.

* Review

In dieser Spalte werden alle Aufgaben abgelegt, die bereits bearbeitet wurden und de­ren Er­gebnis durch eine verantwortliche Person fachlich abgenommen werden muss. In der App­­li­ka­tion Taskitory muss diese Spalte nicht auf einem Kanbanboard sichtbar sein. Die Spalte kann ein- und ausgeblendet werden. Dabei werden die in dieser Spalte be­­find­lichen Auf­ga­ben nicht entfernt. Allerdings können Aufgaben in einer aus­ge­blen­de­­ten Spalte nicht ver­waltet werden.

* Testing

Diese Spalte enthält alle Aufgaben, die bereits bearbeitet wurden und noch technischen Tests un­ter­­zo­gen werden. In der Applikation Tas­ki­tory muss diese Spalte nicht auf ei­nem Kann­ban­­board sichtbar sein. Die Spalte kann ein- und ausgeblendet werden. Da­bei wer­den die in diesen Spalten befindlichen Aufgaben nicht entfernt. Allerdings können Auf­ga­ben in ei­ner ausgeblendeten Spalte nicht ver­wal­tet werden.

* Done

Diese Spalte enthält alle Aufgaben, die abgeschlossen wurden. In der Applikation Tas­ki­­to­ry muss diese Spalte immer auf einem Kanbanboard vorhanden sein.

1. Sprint

Der Begriff „Sprint“ entstammt den agilen Vorgehensmodellen. Sprints sind feste Zeit­räu­me, in die ein Projekt eingeteilt wird. In diesen Zeiträumen nimmt sich das Projekt-Team be­stimmte Aufgaben vor, die mit Hilfe eines Kanban Boards verwaltet werden können. Nach Beendigung eines Sprints werden die Arbeitsergebnisse Qualitätsgesichert. Auf die­se Weise wird ein Produkt iterativ weiterentwickelt und kann mit den Anforderungen ab­ge­glichen werden.

1. Statistiken

Es sollen einfache statistische Analysen auf Kanbanboards durchgeführt werden können. Ein Beispiel ist die Berechnung des Anteils der Aufgaben auf ei­nem Kanban Board, der zum Ende des Sprints in einem be­­stimm­ten Bearbeitungs-Sta­­tus ist.

1. Tag

Ein Tag ist ein Etikett, das einer Aufgabe angeheftet werden kann, um diese Aufgabe mit Me­­ta-Informationen zu versehen. Tags bestehen aus einer global eindeutigen Be­zeich­nung und einer nicht eindeutigen Beschreibung. Tags sind seinem Erzeuger (Benutzer) zu­geordnet.

1. Nachricht

Eine Nachricht dient der Übermittlung von für einen Benutzer relevanten Informationen zu einem Projekt, dem er angehört. Solche relevanten Informationen sind z. B. der Raus­wurf aus einem Projekt, die Änderung der eigenen Rolle oder das Löschen des Pro­jekts.

Nachrichten werden nicht von einem Benutzer verschickt und können daher nicht dazu ge­­nutzt werden, um eine Konversation zu führen. Nachrichten werden von einem Projekt ver­­sendet und von einem Benutzer empfangen. Der Inhalt von Nachrichten kann nicht per­­so­na­lisiert werden und ist für den jeweiligen Anwendungsfall vorbereitet. Über ein Attribut kön­nen Nachrichten als „gelesen“ markiert werden.

### Begriffe der technischen Implementierung

Die Ubiquitous Language der Applikation Taskitory umfasst die folgenden Be­griffe für Objekte, Konzepte und Regeln der technischen Implementierung.

1. Backend

Der Begriff „Backend“ meint im Umfeld der Applikation Taskitory die Java-Anwendung, in der die Geschäftslogik, die Datenbank- und IAM-Anbindung sowie die REST-Schnitt­stel­le der Applikation Taskitory implementiert ist. Das steht im Widerspruch zu der ei­gent­li­chen Bedeutung des Begriffs, da Benutzer direkt auf das Backend zugreifen. Alle anderen Eigenschaften sind jedoch typisch für ein Backend. Wenn zukünftig die Applikation Tas­ki­to­ry durch ein Frontend mit GUI erweitert wird, kann der direkte Zugriff der Benutzer auf das Back­end verhindert werden. Dann handelt es sich um ein echtes Backend.

1. Universally Unique Identifier (UUID)

„Universally Unique Identifier“ (UUID) sind 128-Bit-Zahlen, die für die Identifikation von Ob­jekten besonders in Informationstechnik-Systemen verwendet werden. UUIDs gelten als praktisch global eindeutig, da die Anzahl möglicher UUIDs sehr groß ist.

1. String

„String“ ist ein komplexer Datentyp von Java. Variablen von diesem Typ können mit end­li­chen un­ter­schiedlich langen Zeichenketten befüllt werden. Strings werden dazu ver­wen­det Text zu repräsentieren.

1. Final (Unveränderlichkeit)

Bestimmte Strukturen in Java wie z. B. Variablen können mit dem Schlüsselwort „final“ de­klariert werden. Dadurch können die Werte der Variablen nach der Initialisierung nicht mehr verändert werden.

1. Zeitstempel

Zeitstempel ist ein komplexer Datentyp für die Repräsentation eines Zeitpunkts. Dabei wird i. d. R. die Zeitspanne seit einem bestimmten Zeitpunkt – oft 01.01.1970 00:00 – ge­spei­chert. Der Wert des Zeitstempels ergibt sich durch die Addition der Zeitspanne zu die­sem Ausgangspunkt. Der Wert kann in verschiedenen Formaten dargestellt wer­den.

1. Representational State Transfer (REST)

Representational State Transfer (REST) ist ein Konzept für die Architektur von verteilten Soft­ware-Systemen. Ein REST-Service stellt Hypermedia-Ressourcen über HTTP im In­ter­net zur Verfügung. Hypermedia-Ressourcen sind Informationen, die nicht auf ein Medium fest­gelegt sind und sich aus mehreren per Hyperlink verbundenen Daten-Objekten zu­sam­mensetzen. Eine solche Ressource wird über eine global eindeutige Adresse und eine Zu­griffsmethode (HTTP-Verben) zugänglich gemacht.

1. HTTP

Das Hypertext Transfer Protocol ist ein zustandsloses Protokoll zur Übermittlung von Da­ten in einem Netzwerk. HTTP wird für den Zugriff auf die REST-API der Applikation Tas­­kitory und innerhalb für die Kommunikation der Applikationskomponenten ver­wen­det.

1. Java Persistence API (JPA)

Die „Java Persistence API“ (JPA) ist ein de fakto Standard für die Integration verschiedener Da­­ten­banken in eine Java-Applikation. Durch das Konzept vom Objektrelationalen Map­ping (OR-Mapping) können in einer Applikation erzeugte Objekte ohne großen Pro­gram­mieraufwand direkt in einer Datenbank persistiert werden.

1. Objektrelationales Mapping

Objektrelationales Mapping bezeichnet eine Technologie, mit der die Objekt-Struktur ei­ner objektorientierten Programmiersprache auf die Relationen-Struktur eines re­la­tio­na­len Datenbank-Systems abgebildet wird. Dabei werden Tabellen mit Spalten durch Klassen mit Attributen repräsentiert. Die Datensätze entsprechen im OR-Mapping den In­stan­zen der Klassen.

1. IAM-/Keycloak-Server

Unter Identity und Access Management (IAM) werden allgemein die Techniken ver­stan­den, mit denen Benutzer einer Software-Applikation authentifiziert und autorisiert wer­den. Das umfasst die Konfiguration von Authentifizierungs- sowie Au­to­ri­sie­rungs­me­tho­den, Rechtegruppen uvm.

Ein IAM-Server ist im Kontext der Applikation Taskitory einer Anwendung, die die­se IAM-Funktionen für eine andere Applikation übernimmt. Für die Applikation Tas­ki­to­ry wird dabei konkret die Applikation „Keycloak“ verwendet. Diese Applikation spei­chert ihre Da­ten in der gleichen Datenbank, in der auch das Backend Daten persistiert.

1. Datenbank/Postgresql

Eine Datenbank ist eine Software-Anwendung, die für die persistente Speicherung von Da­ten verwendet wird. Die Daten werden dabei hauptsächlich von anderen An­wen­dun­gen zur Verfügung gestellt. Mit der Structured Query Language (SQL) können die Da­ten­be­stände ei­ner DB bearbeitet werden. Für die Applikaiton Taskitory wird eine Post­gresql-DB verwendet.

1. Docker-Container

Docker ist eine Software-Applikation, mit der andere Software-Applikationen in „Con­tai­nern“ betrieben werden können. Diese Container sind virtualisierte Laufzeitumgebungen für Soft­­ware-Applikation, die mit den Ressourcen eines Gastgeber-Systems betrieben wer­den. Mit „Dockerfiles“ können diese Laufzeitumgebungen kon­fi­gu­riert und auf ver­schie­denen Computern exakt wiederhergestellt wer­­den. Dadurch können gan­ze App­li­ka­tio­nen schnell auf andere Server bewegt und wie­­der hergestellt werden. Um den Zustand ei­ner Applikation zu speichern und in einem neu­en Container wieder her­zustellen, wer­den „Volumes“ verwendet. Volumes sind Verzeichnisse des Gastgeber-Systems, die in einen Container gemounted werden können. Die Verzeichnisse des Gastgeber-Systems sind unabhängig von den Containern und werden daher nicht durch das Löschen einer Con­tainer-Instanz geleert. Der Zustand einer Applikation bleibt in solchen Volumes für neue Container-Instanzen erhalten.

1. Docker Compose

Docker-Compose ist eine Technologie, mit der mehrere Docker-Container zusammen ver­wal­tet wer­den können. Das umfasst die Konfiguration der einzelnen Container, aber auch die Kommunikationswege der Container untereinander und nach au­ßen. Auf diese Weise kön­nen mehrere Container zu einem System ver­bun­den werden. Mit einer „Do­cker-Com­pose.yml“ Datei können solche Systeme beschrieben werden.

Durch einbinden einer „.env“ Datei ist es möglich sensible Informationen wie Passwörter aus dem Docker-Compose.yml und anderen Applikations-internen Dateien in Um­ge­bungs­­­variablen auszulagern.

1. Postman Collection

Postman ist eine Software-Applikation mit der HTTP-Anfragen gezielt konfiguriert, ver­sen­det und empfangen werden können. Eine „Collection“ ist eine Sammlung von HTTP-An­fragen, die mit einer Verzeichnis-Struktur verwaltet und durch übergreifende Kon­fi­gu­ra­tionen wie z. B. die Authentifizierung von HTTP-Anfragen erweitert werden. Für die App­­likation Taskitory wird eine solche Collection mit zentralen HTTP-Anfragen erstellt.

1. Hashing

Beim Hashen einer Zeichenfolge wird eine Einweg-Hash-Funktion auf diese Zeichenfolge an­gewendet. Hash-Funktionen bilden Zeichenfolgen beliebiger Länge auf Zeichenfolgen mit einer festen Länge ab. Der erzeugte Hash-Wert einer Hash-Funktion darf praktisch nicht auf sei­nen Ausgangswert zurückführbar sein. Zudem darf es in annehmbarer Zeit nicht mög­lich sein, zwei Zeichenketten zu finden, die den gleichen Hash-Wert haben.

## Taktische Muster des Domain Driven Design

Die taktischen Muster des Domain Driven Desings werden dazu verwendet ein möglichst präzises und gut verständliches Modell der Fachlichkeit einer Applikation zu erstellen. Dafür werden fünf Muster definiert, die unterschiedliche Aufgabenbereich haben. Nach­fol­gend wird auf die fünf taktischen Muster des Domain Driven Designs mit ihren Auf­ga­benbereichen und der Umsetzung in der Applikation Taskitory eingegangen.

### Entitäten

Entitäten sind Objekte der Domäne, die nicht durch ihre Attribute, sondern durch ihre Iden­ti­tät unterschieden werden können. Die Identität wird dabei i. d. R. durch ein Schlüssel-At­tribut dargestellt.

In der Problemdomäne der Applikation Taskitory interagieren die Ak­teure Projekt, Projekt-Mitgliedschaft, Nach­­richt, Kanbanboard, Auf­ga­be, Be­nut­zer, Tag. Daher sind all diese Akteure Kandidaten für Domänen-Entitäten. Die benannten Akteure ent­hal­ten wichtige Informationen, die in der DB persistiert werden müs­sen. Dazu werden die Ak­teure mit einem JPA-OR-Mapping versehen. Dadurch enthalten alle Akteure ohnehin ei­nen DB-Pri­mär­schlüssel. Allerdings muss das nicht heißen, dass ein Akteur auch eine En­ti­tät ist. Daher wird nachfolgend überprüft, welche Akteure auch Entitäten sind.

1. Projekt

Projekte werden durch eine eindeutige Bezeichnung identifiziert. Diese Bezeichnung ist der Primärschlüssel der zugehörigen DB-Entität. Es handelt sich um einen natürlichen Schlüs­sel, der die Identität eines Projekts repräsentiert. Die Beschreibung und das Pro­jekt-Team eines Projekts repräsentieren nicht die Identität eines Projekts. Es kann meh­re­re Projekte mit gleichen Werten für diese Attribute geben.

Der geheime Projekt-Schlüssel ist ein zusätzlicher Schlüsselkandidat, da dieser Projekt-Schlüs­sel praktisch global eindeutig ist. Allerdings ist eine Dopplung theoretisch möglich, da der Projekt-Schlüssel zufällig generiert wird. Die Wahrscheinlichkeit, dass eine Dopp­lung auftritt, ist bei 36128 möglichen Projekt-Schlüs­seln allerdings marginal.

Projekte bestehen neben beschriebenen Attributen wie der Bezeichnung und einer Beschreibung aus Kanban Boards, Aufgaben, Nachrichten und Projekt-Mitgliedschaften. Die­se anderen Akteure sind unveränderbar diesem einen Projekt zugeordnet und können in keinem anderen Projekt auftreten. Durch die Kanban Boards, Aufgaben, Nachrichten und Projekt-Mitgliedschaften könnte ein Projekt identifiziert werden.

Allerdings sind alle Attribute eines Projekts mit Ausnahme der Bezeichnung veränderbar und re­prä­sen­­tieren den Lebenszyklus dieses Projekts. Daher eignet sich die Bezeichnung am besten als Primär-Schlüssel

Projekte unterscheiden sich durch ihre Identität in Form einer eindeutigen Bezeichnung. Zu­dem unterliegen Projekte einer ständigen Veränderung. Daher ist ein Projekt eine Do­mä­nen-Entität.

1. Projekt-Mitgliedschaft

Eine Projekt-Mitgliedschaft verbindet ein Projekt mit einem Benutzer. In der DB wird eine Pro­jekt-Mitgliedschaft durch eine n:m-Beziehung abgebildet. Eine Projekt-Mitgliedschaft wird daher eindeutig durch eine Kombination von Projekt-Bezeichnung und Benutzer-ID iden­tifiziert. Benutzer sollen maximal eine Mitgliedschaft in einem Projekt besitzen. Es soll nicht ge­spei­chert werden, ob und wie oft ein Benutzer einem Projekt beigetreten ist und es wieder ver­lassen hat. Daher wird der Zeitstempel des Starts der Mitgliedschaft nicht als Teil des Pri­mär­schlüs­sels verwendet.

Der Start der Mitgliedschaft wird neu gesetzt, wenn der Benutzer befördert wird. Eine Beförderung meint das Ändern der zugeordneten Rolle von „Mitglied“ zu „Admin“. Projekt-Mitgliedschaften bestehen also aus einem zusammengesetzten Primärschlüssel (Projekt-Bezeichnung, User-ID) der die Identität der Projekt-Mitgliedschaft ausmacht und zwei beschreibenden Attributen, die während des Lebenszyklus einer Projekt-Mitgliedschaft verändert werden können.

Projekt-Mitgliedschaften unterscheiden sich durch ihre Identität in Form einer Zu­ord­nung eines Benutzers zu einem Projekt. Zu­dem können Projekt-Mitgliedschaften über die Zeit verändert werden. Daher ist eine Projekt-Mitgliedschaft eine Do­mä­nen-Entität.

1. Nach­­richt

Nachrichten haben einen Absender und einen Empfänger. Der Absender kann bei der App­likation Taskitory nur ein Projekt sein. Der Empfänger ist immer ein Benutzer. Auch die­ser Akteur verbindet die Akteure Projekt und Benutzer. Allerdings werden Nach­rich­ten nicht durch ihre Projekt- und Benutzer-Zugehörigkeit identifiziert. Es kann viele ver­schie­dene Nachrichten geben, die von dem gleichen Projekt an den gleichen User ver­schickt werden.

Nachrichten haben einen Typ und einen Inhalt, die zusammengenommen die Bedeutung ei­ner Nachricht ausmachen. Allerdings können auch diese Attribute nicht zur Iden­ti­fi­zie­rung einer Nachricht verwendet werden. Denn es können mehrere Nachrichten des glei­chen Typs mit dem gleichen Inhalt von dem gleichen Projekt an den gleichen Benutzer ver­schickt werden.

Durch Hinzufügen des Empfangszeitpunkts einer Nachricht zu den bisher betrachteten Att­ributen könnte ein Primärschlüssel entstehen. Praktisch scheint es sehr un­wahr­schein­lich, dass zwei Nachricht mit gleichen Werten für diese fünf Parameter vorliegen. Theo­retisch ist das allerdings kein Problem.

Abschließend bleibt noch das Attribut, mit dem eine Nachricht als gelesen markiert wird. Die­ses Attribut liefert keine Möglichkeit zur Identifizierung einer Nachricht. Hinzu kommt, dass sich dieser Status einer Nachricht während des Lebenszyklus einer Nach­richt ändern kann. Alle anderen Eigenschaften einer Nachricht können nicht verändert wer­den.

Die geschilderten Eigenschaften von Nachrichten, sprechen nur eingeschränkt dafür, dass Nach­­richten Domänen-Entitäten sind. Dass nahezu alle Attribute einer Nach­richt un­ver­än­der­lich und notwendig sind, um eine Nachricht zu identifizieren, spricht eher für ein Value Object. Allerdings können Nachrichten zur Laufzeit mit dem At­tri­but „read“ (zu Deutsch „gelesen“) verändert werden. Zudem ist dieses Attribut nicht re­levant für die Identifizierung von Nachrichten. Daher werden Nachrichten in der Applikation Taskitory als Entität definiert. Um Nachrichten zu identifizieren wird eine UUID (Sur­rogatschlüssel) als Primärschlüssel-Attribut verwendet.

1. Kanban Board

Kanban Boards besitzen eine Menge von beschreibenden Attributen. Dazu zählen eine für ein Projekt eindeutige Bezeichnung, eine Beschreibung, ein Sprint-Zeitraum sowie zwei At­tribute für das Ein- und Ausblenden von der Review- und Testing-Spalte eines Kanban Boards. Alle diese Attribute können allein nicht zur Identifizierung von Kanban Boards verwendet werden.

Kanban Boards sind immer einem Projekt unveränderlich zugeordnet. Dieses Projekt kann zusammen mit der Bezeichnung zur Identifizierung eines Kanban Boards verwendet wer­den. Allerdings kann die Bezeichnung eines Kanban Boards genau wie die anderen be­schreibenden Attribute verändert werden.

Eine weitere Möglichkeit zur Identifizierung eines Kanban Boards könnte die Sammlung der dem Kanban Board zugewiesenen Aufgaben sein. Allerdings kann auch diese Samm­lung von Aufgaben sich verändern.

Letzt­lich bleibt nur noch die Möglichkeit zur Identifizierung eines Kanban Boards mittels ei­nes Surrogatschlüssels in Form einer UUID. Da die Identität eines Kanban Boards von ei­nem solchen Schlüssel abhängt und Kanban Boards einen Lebenszyklus mit Verän­de­run­gen durchläuft, werden Kanban Boards als Entität definiert.

1. Auf­ga­be

Aufgaben sind die zentralen Träger von Informationen in der Applikation Taskitory und bestehen daher zu einem Großteil aus beschreibenden Attributen. Dazu zählen eine für ein Projekt eindeutige Bezeichnung, eine Beschreibung sowie zwei Attribute für die Bemessung der Komplexität und Priorität einer Aufgabe. Weiterhin wird der Bearbeitungs-Zeitraum, der Autor und der für die Bearbeitung zuständige Benutzer festgehalten. Alle diese Attribute enthalten Werte, die nicht spezifisch für eine einzelne Aufgabe sind.

Höchstens die Bezeichnung der Aufgabe liefert in Kombination mit dem Projekt, zu dem die Aufgabe gehört, eine Möglichkeit zur Identifizierung einer Aufgabe. Doch wie schon bei den Kanban Boards, können auch bei den Aufgaben alle beschreibenden Attribute im Lebenszyklus einer Aufgabe verändert werden. Ähnlich verhält es sich mit einer Kombination aus dem Kanban Board und dem Projekt, dem eine Aufgabe zugewiesen ist, sowie der Bezeichnung einer Aufgabe.

Auch über die Sammlung der einer Aufgabe zugewiesenen Tags, kann diese Aufgabe nicht iden­tifiziert werden. Abschließend bleibt auch hier wieder ein Surrogat-Schlüssel in Form ei­ner UUID. Bei dem Akteur „Aufgabe“ können die gleichen Schlüsse wie schon beim Ak­teur „Kanban Board“ gezogen werden. Aufgaben werden daher als Entität definiert.

1. Benutzer

Benutzer sind die Repräsentationen von Menschen. Menschen können nicht durch At­tri­bu­te wie den Vor- und den Nachnamen allgemein identifiziert werden. Die Be­nut­zer­kon­ten dieser Menschen können allerdings oft durch Attribute wie eine E-Mail-Adresse oder ei­nen Benutzernamen identifiziert werden. In der Applikation Taskitory wird allerdings ein von Keycloak bereitgestellter Surrogatschlüssel verwendet. Da die gesamte DB-Ta­bel­le von Keycloak bereitgestellt und verwaltet wird, kann dies nicht verändert werden.

Möglich wäre eine Identifizierung auch über die erstellten Aufgaben und Tags. Da sich diese Sammlung von anderen Akteuren verändern kann, ist das allerdings schlechter als die von Keycloak verwendete Methode.

Mit Ausnahme der ID (Identifikationsnummer) eines Benutzers können alle von Keycloak vor­gegebenen Attribute eines Benutzers geändert werden. Das wird allerdings durch die Funk­tionen von Keycloak und nicht durch eigene Operationen auf den DB-Tabellen des Key­cloak umgesetzt. Auch die Verbindungen zu anderen Akteuren (z. B. erstellte Auf­ga­ben) verändern sich während des Lebenszyklus eines Benutzers. Daher kann der Akteur „Be­nutzer“ als Entität definiert werden.

1. Tag

Tags bestehen lediglich aus einer Bezeichnung, einer Beschreibung und einer Referenz auf den Erzeuger. In diesem Fall muss allerdings kein Surrogatschlüssel ver­wendet wer­den, da die Bezeichnung eines Tags global eindeutig ist. Die Bezeichnung ist damit ein na­türlicher Schlüssel.

Die Bezeichnung und die Referenz auf den Erzeuger eines Tags sind unveränderlich. Allerdings kann die Beschreibung eines Tags während des Lebenszyklus eines Tags verändert werden.

Da die Identität eines Tags eindeutig durch einen Schlüssel repräsentiert wird und Tags ver­änderbare Attribute beinhalten, wird der Akteur „Tag“ als Entität definiert. Wenn ein At­tribut keine Beschreibung hätte, wäre der Akteur „Tag“ ein Value Object.

### Value Objects

Value Objects Modellieren die Elemente der Problemdomäne, die keine inhärente Iden­ti­tät besitzen. Die Identität von Value Objects wird durch die Werte aller Attribute des Value Ob­jects repräsentiert. Dadurch können zwei Java-Objekte dasselbe Value Object re­prä­sen­­tieren. Value Objects sind unveränderlich. Value Objects kapseln in sich. In Entitäten wer­den Value Objects dazu verwendet, um geschlossene Subaufgabenbereiche zu kapseln und dadurch die Komplexität der Entitäten zu reduzieren.

Auf der Domänenschicht der Applikation Taskitory werden Value Objects ausschließlich in Entitäten eingebettet verwendet. Nachfolgend werden die Value Objects be­schrie­ben.

1. Projekt-Schlüssel

Der Projekt-Schlüssel besteht aus einem einzelnen unveränderlichen Attribut, das die global eindeutige Zeichenfolge – den geheimen Projekt-Schlüssel – enthält. Das einzige Attribut des Value Objects ist unveränderlich und stellt selbst den Projekt-Schlüssel da.

Das Value Object „Projekt-Schlüssel“ ist als Attribut in der Projekt Entität eingebettet. Da­durch wird die Erzeugung und Bereitstellung des Projekt-Schlüssels als Hash von dem Rest der Projekt Entität abgeschottet.

1. Nachrichten-Inhalt

Auch der Nachrichten-Inhalt besteht nur aus einem einzelnen Attribut. In diesem Attribut wird der Inhalt einer Nachricht als String gespeichert. Dieses Attribut wurde nicht als „final“ definiert, da JPA einen Konstruktor ohne Parameter für Bestandteile des OR-Map­pings for­dert.

Da es keinen sinn­vol­len Standard-Wert für den Nachrichten-Inhalt gibt, muss die Un­ver­än­derlichkeit dieses At­tributs auf anderem Wege erreicht werden. Um zu verhindern, dass der Konstruktor oh­ne Attribute von der Domänen-Logik verwendet werden kann, wird die Sichtbarkeit die­ses Konstruktors eingeschränkt. Zusätzlich gibt es keine Me­tho­den, mit denen das At­tri­but nach seiner Initialisierung auf einen anderen Wert gesetzt wer­den kann. Dadurch ist das Attribut effektiv unveränderlich.

Das Value Object „Nachrichten-Inhalt“ ist als Attribut in der Nachrichten Entität ein­ge­bet­tet. Dadurch wird das Umwandeln des Nachrichten-Inhalts von String zu JSON und um­ge­kehrt von dem Rest der Nachriten Entität abgeschottet.

1. Sprint

Ein Sprint ist ein Zeitraum und wird in der Applikation Taskitory jeweils durch einen Zeit­stem­pel für den Start und für das Ende des Sprints definiert. Beide Attribute sind un­ver­än­derlich und realisieren zusammen einen Sprint. Ein Attribut allein reicht dafür nicht aus. Beide Attribute formen zusammen die Identität des Sprints.

Der Standardwert für den Startzeitpunkt ist der Zeitpunkt der Erstellung des Objekts und der Endzeitpunkt exakt 2 Wochen darauf. Diese zwei Wochen sind ein gängiger Wert für einen Sprint-Zeitraum und können daher als Standardwert verwendet werden.

Das Value Object „Sprint“ ist als Attribut in der Kanban Board Entität ein­ge­bet­tet und schottet dadurch die Validierung der Zeitstempel und die Prüfung, ob der Sprint vorbei ist von der Entität ab.

1. (Aufgaben-)Komplexität

Die Komplexität einer Aufgabe wird durch ein einzelnes Attribut realisiert. Dieses At­tri­but hält in Form einer Zahl im Bereich [0, 20] den Grad der Komplexität fest. Dabei spre­chen höhere Werte für eine größere Komplexität. Das At­tribut ist un­veränderlich und be­nö­tigt daher eine Standard-Initialisierung. Als Stan­dard-Wert wird in diesem Fall der „leere“ Wert 0 verwendet.

Das Value Object „Komplexität“ ist als Attribut in der Aufgaben Entität ein­ge­bet­tet und schottet dadurch die Validierung der Komplexitätswerte von der Entität ab.

1. (Aufgaben-)Priorität

Die Priorität einer Aufgabe wird durch ein einzelnes Attribut realisiert. Dieses At­tri­but hält in Form einer Zahl im Bereich [0, 20] die Priorität fest. Ein geringerer Wert bedeutet ei­ne höhere Priorität. Auch die­ses At­tribut ist unveränderlich und benötigt eine Stan­dard-Ini­tialisierung. Als Stan­dard-Wert wird in diesem Fall der „leere“ Wert 0 ver­wen­det.

Das Value Object „Priorität“ ist als Attribut in der Aufgaben-Entität ein­ge­bet­tet und schottet dadurch die Validierung der Prioritätswerte von der Entität ab.

1. Aufgaben-Lebenszyklus

Der fachliche Lebenszyklus einer Aufgabe in der Problemdomäne der Applikation Tas­ki­to­ry wird jeweils durch einen Zeit­stem­pel für den Start und für das Ende des Be­ar­bei­tungs­zeitraums sowie einen Status definiert. Dieser Status wird verwendet, um Aufgaben auf einem Kanban Board einzuordnen. Der Startzeitpunkt eines Aufgaben-Lebenszyklus ist unveränderlich. Dieser Wert wird mit dem Zeitpunkt der Erzeugung des Objekts initialisiert.

Die beiden Attribute für den Status und den Endzeitpunkt sind allerdings veränderlich, um den Bearbeitungs-Prozess einer Aufgabe abzubilden. Besonders der Endzeitpunkt kann nicht bei der Erzeugung des Value Objects initialisiert werden, da der Endzeitpunkt noch ungewiss ist. Allerdings ist das nicht der einzige Fall, in dem ein Attribut verändert werden muss. Daher kann der Aufgaben-Lebenszyklus nicht als reines Value Object definiert werden. Allerdings handelt es sich auch nicht um eine Entität. Denn die Identität eines Aufgaben-Lebenszyklus wird durch alle drei Parameter-Werte definiert und nicht durch eine ID. Hinzu kommt die starke Bindung ausschließlich an die Aufgaben Entität.

Der Value Object Aufgaben-Lebenszyklus ist das Pendant zu der Tag Entität. Denn in bei­den Fällen passt eine Definition als Entität oder Value Object nur mittelmäßig. Das Val­ue Ob­ject „Aufgaben-Lebens-Zyklus“ ist als Attribut in der Aufgaben Entität ein­ge­bet­tet und schot­tet dadurch die Zustandsänderung einer Aufgabe von den anderen Funk­tio­nen des Auf­gaben Entität ab.

1. Irrelevante Benutzerdaten

Wie bereits erwähnt, wird die DB-Tabelle der Benutzer Entität durch die Applikation Key­cloak verwaltet. Auf die Benutzer Entität soll daher nur lesend zugegriffen werden kön­nen. Zudem sollen nur der Benutzername und die ID des Benutzers gelesen werden können. Die anderen von Keycloak verwendeten Attribute sind für diese Applikation nicht rele­vant. Der Zugriff auf diese Attribute wird daher vollständig verwehrt. Um dies effizient und übersichtlich umzusetzen wird dieses Value Object definiert.

Alle Attribute des Value Objects „Irrelevante Benutzerdaten” können nach der Ini­tia­li­sie­rung nicht geändert werden. Theoretisch kann eine Identität über ein Schlüssel-Attribut wie z. B. die E-Mail-Adresse festgestellt werden. Praktisch lässt sich allerdings schwierig sa­gen, ob diese Annahme stimmt, da diese Tabelle von Keycloak verwaltet wird. Es bleibt zu beachten, dass dieses Va­lue Object nie ohne ei­ne Benutzer-Instanz verwendet wird und nur der Effizienz und Über­sichtlichkeit halber ver­wendet wird. Man könnte von einer Art Schein Value Object spre­chen.

1. Bezeichnung

Dieses Value Object wird übergreifend definiert, da fast alle Entitäten eine Bezeichnung mit den gleichen Eigenschaften und Restriktionen haben. Mit diesem Value Object wird si­chergestellt, dass keine leeren oder zu langen Bezeichnungen vergeben werden. Das Value Object besteht lediglich aus einem Attribut für die Bezeichnung selbst. Es wird keine ID benötigt, da das Attribut das Objekt selbst repräsentiert.

1. Beschreibung

Dieses Value Object wird übergreifend definiert, da fast alle Entitäten eine Beschreibung mit den gleichen Eigenschaften und Restriktionen haben. Mit diesem Value Object wird si­­chergestellt, dass keine leeren oder zu langen Beschreibungen vergeben werden. Das Val­­ue Object besteht lediglich aus einem Attribut für die Beschreibung selbst. Es wird kei­ne ID benötigt, da das Attribut das Objekt selbst repräsentiert.

### Aggregate

„Aggregate“ fassen Entitäten und Value Objects zu transaktionalen Einheiten zusammen. Ag­gregate werden über einen „Aggregate Root“ verwaltet, der die Einhaltung der Do­mä­nen-Regeln des Aggregats sicherstellt. In der Applikation Taskitory werden das „Projekt Ag­gregat“ und das „Benutzer Aggregat“ verwen­det. Nach­fol­gend werden beide Aggregate nä­her beschrieben.

1. Projekt Aggregat

Das erste Aggregat verwendet die Projekt Entität als Aggregate Root. Zudem beinhaltet das Projekt Aggregat die Kanban Board, Aufgaben, Nachrichten und Projekt-Mitglied­schafts En­ti­tä­ten. Die Value Objects „Projekt-Schlüssel“, „Komplexität“, „Priorität“, „Auf­ga­ben Lebenszyklus“, „Nachrichten-Inhalt“ und „Sprint" sind ebenfalls Teil dieses Aggre­gats, da sich die Bezugs-Entitäten in diesem Aggregat befinden.

Die Kanban Board Entität wurde dem Projekt Aggregat zugeordnet, da Kanban Boards nur von der Projekt Entität abhängig sind und nur über ein Projekt erreichbar sein sollen. Das heißt, dass die Funktionen von Kanban Boards nur über ein Projekt aufgerufen wer­den dürfen. Dabei klärt die Projekt Entität alle Rahmenbedingungen (z. B. gehört das Kanban Board zu diesem Projekt) und gibt den Funktionsaufruf dann an das Kanban Board weiter.

Die Aufgaben Entität wurde dem Projekt Aggregat zugeordnet, da Aufgaben hauptsächlich von der Projekt und der Kanban Board Entität abhängig sind. Allerdings besteht über das Au­tor-Attribut auch eine zwingende Abhängigkeit von der User Entität. Trotzdem werden die Aufgaben dem Projekt Aggregat zugeordnet. Denn Aufgaben sollen nur über ein Pro­jekt erreichbar sein. User können keine Aufgaben ohne Projekt erstellen. Projekte können für eine Benutzer-Referenz die in dem gleichen Aggregat enthaltenen Projekt-Mit­glied­schaf­ten befragen. Die Funktionen von Aufgaben dürfen indirekt über ein Kanban Board und direkt über ein Projekt aufgerufen wer­den. In jedem Fall klärt die Projekt Entität alle Rah­menbedingungen für die Verwendung einer Aufgabe.

Die Nachrichten sowie die Projekt-Mitgliedschafts Entität stellen Beide Verbindungen zwischen der Benutzer Entität und der Projekt Entität her. Die Nachrichten und die Projekt-Mitgliedschafts Entität könnten daher dem Projekt- oder dem Benutzer Aggregat zugeordnet werden. Für die Applikation Taskitory wurde in beiden Fällen entschieden, die Entitäten dem Projekt Aggregat zu zuordnen, da die Funktionen der beiden Entitäten im­mer von einem Projekt angestoßen werden. Die initiale Interaktion geht immer von dem Projekt aus. Daher muss der Zugriff auf die Funktionen der Entitäten über die Projekt En­tität ablaufen.

1. Benutzer Aggregat

Das Benutzer Aggregat beinhaltet die Benutzer und die Tag Entität inklusive anhängiger Value Objects (Irrelevante Benutzerdaten). Die Benutzer Entität ist nicht Teil des Projekt Aggregats, da die Benutzer-Instanzen unabhängig von einem Projekt bestehen können. Die Tag Entität kann dies ebenfalls. Zusätzlich ist die Tag Entität ausschließlich von der Benutzer Entität abhängig. Dadurch wird die Benutzer Entität zum Aggregate Root, die die Zugriffe auf die Tag Entität regelt.

### Repositories

Repositories dienen als Schnittstelle der Domänenschicht zum Persistenz-Framework auf der Plugin-Schicht. Die Repositiories geben vor, mit welchen Funktionen auf die persistenten DB-Einträge zugegriffen werden darf. Es sollte im Allgemeinen für jedes Aggregat ein Repository verwendet werden, sodass Aggregate immer als Einheit geladen und persistiert werden. Die Applikation Taskitory verwendet entsprechend je ein Re­po­si­tory für das Projekt und das Benutzer Aggregat.

### Domänen Services

Domänen Services werden verwendet, um Entitäten übergreifende Logik oder Funk­tio­na­litäten zu implementieren. Die Applikation Taskitory benötigt auf der Domänen-Schicht einen „RessourceAccessService“ mit dem überprüft wird, ob ein Benutzer auf be­stimm­te Ressourcen eines Projekts zugreifen kann. Diese Funktionen können keinem der bei­den Aggregate allein zugeordnet werden, da sie in beiden Aggregaten eingesetzt wer­den und sich immer auf Entitäten aus beiden Aggregaten beziehen.

# Use Cases

In diesem Kapitel werden zentrale Use Cases der für die Applikation Taskitory the­ma­tisch sortiert aufgeführt. Die hier geschilderten Informationen sind be­son­ders für die App­li­ka­tions-Schicht der Clean Architecture relevant.

## Projekte

1. Projekt anlegen

Ein Projekt wird von einem Benutzer mit einer Be­zeichnung und einer Beschreibung an­ge­legt. Der geheime Gruppen-Schlüssel wird automatisch generiert. Optional kann der Be­nut­zer direkt ein Projekt-Team aus Benutzern zusammenstellen. Dabei kann er ent­schei­den, wel­che Rolle ein Benutzer-Konto erhält. Der Ersteller eines Projekts wird au­to­ma­tisch als Ad­mi­nistrator dem Projekt zugeordnet. Ein Projekt beinhaltet nach der Ini­tia­li­sie­rung keine Aufgaben und keine Kanban Boards.

1. Projekt-Attribute anpassen

Die Beschreibung und der geheime Projekt-Schlüssel eines Projekts sollen von einem Pro­jekt-Ad­mi­nis­tra­tor geändert werden können. Benutzer mit der Rolle „Mitglied“ können die­­se At­tri­bute nicht ändern. Der geheime Schlüssel eines Projekts kann nur auf einen zu­fälligen Wert geändert werden.

1. Projekt löschen

Das Löschen eines Projekts bewirkt, dass alle damit verbundenen Kanban Boards, Auf­ga­ben, Nachrichten und Projekt-Mitgliedschaften gelöscht werden. Vor dem Löschen all die­ser Verbindungen werden die Benutzer über das Löschen des Projekts per Nachricht in­for­miert.

## Projekt-Mitgliedschaften

1. Benutzer-Konto zu einem Projekt hinzufügen

Das Projekt-Team kann auch nach der Initialisierung erweitert werden. Um einem Projekt beitreten zu können, muss ein Benutzer den geheimen Schlüssel eines Projekts angeben. Diesen geheimen Schlüssel können Projekt-Administratoren per Nachricht an den jeweiligen Benutzer schicken. Damit der Schlüssel nicht so einfach kompromittiert werden kann, wird der Schlüssel gehasht übertragen. Wenn ein Benutzer einem Projekt beigetreten ist, werden alle Projekt-Mitglieder und auch der neu hinzugefügt Benutzer darüber in einer Nachricht informiert.

1. Ein Benutzer-Konto befördern

Projekt-Administratoren sollen einfache Mitglieder zu Projekt-Administratoren be­för­dern können. Der beförderte Benutzer und alle Mitglieder eines Projekt-Teams sollen über die Beförderung informiert werden.

1. Benutzer-Konto aus Projekt-Team entfernen

Ein Benutzer kann aus eigenem Antrieb aus einem Projekt austreten. Ein Benutzer mit der Rolle „Mitglied“ kann auch gegen seinen Willen von einem Projekt-Administrator aus einem Projekt entfernt werden. Dadurch werden die Referenzen auf den Benutzer aus dem Projekt entfernt. Der entfernte Benutzer soll per Nachricht informiert werden, wenn ein Administrator ihn aus einem Projekt entfernt hat.

## Kanban Boards

1. Kanban Board anlegen

Kanban Boards können mit einer für ein Projekt eindeutigen Bezeichnung, einer Beschreibung und einem festgelegten Sprint-Zeitraum erzeugt werden. Optional können noch die zwei Spalten „Review“ und „Testing“ aktiviert werden.

Ein Kanban Board ist bei Er­zeu­gung leer und besitzt die oben benannten Spalten. Es sollen mehrere Kanban Boards in einem Projekt gleichzeitig bestehen können.

1. Attribute eines Kanban Boards bearbeiten

Die Bezeichnung und die Beschreibung eines Kanban Boards können durch die Benutzer des entsprechenden Projekts angepasst werden. Dabei bleibt zu beachten, dass die Bezeichnung in einem Projekt eindeutig sein muss.

Außerdem ist es möglich die Spalten „Review“ und „Testing“ auszublenden. Alle in diesen Spalten befindlichen Aufgaben werden daraufhin ausgeblendet. Allerdings behalten sie den Status und können anschließend wieder eingeblendet werden.

1. Kanban Board löschen

Kanban Boards können durch Benutzer des entsprechenden Projekts gelöscht werden. Die darauf befindlichen Aufgaben werden daraufhin in das Backlog gelegt. Die Aufgaben behalten ihren Bearbeitungs-Status.

1. Aufgabe auf einem Kanban Board verschieben

Die zentrale Funktion eines Kanban Boards ist, Aufgaben über ihren Bearbeitungs-Sta­­tus zu verwalten. Be­nut­zer können die Aufgaben auf einem Kanban Board in einen anderen Be­arbeitungs-Status verschieben. Der Bearbeitungs-Status einer Aufgabe im Backlog ei­nes Projekts kann nicht geändert werden.

1. Kanban Board überführen

Das Kanban Board eines vergangenen Sprints kann in ein neues Kanban Board überführt wer­den. Das ist dann interessant, wenn sich auf dem alten Kanban Board noch nicht ab­­ge­schlos­se­ne Aufgaben befinden. Diese Aufgaben sollen in den gleichen Status des neu­en Kanban Boards über­nommen werden.

1. Statistiken erstellen

Es soll möglich sein, Statistiken über ein bestehendes Kanban Board anzulegen. Mögliche Anwendungsfälle sind z. B. die Verteilung von Aufgaben nach Bearbeitungs-Status oder die Berechnung einer durchschnittlichen Bearbeitungsdauer.

## Nachrichten

1. Benutzer zu Projekt einladen

Projekt-Administratoren können Benutzer, die noch kein Teil des Projekt-Teams sind zu dem Projekt einladen. Dazu muss der geheime Schlüssel eines Projekts von einem Projekt-Ad­ministrator an diesen Benutzer in gehashter Form geschickt werden. Der Benutzer muss diesen gehashten Schlüssel vorweisen, um einem Projekt beizutreten.

1. Nachrichten empfangen

Benutzer können die erhaltenen Nachrichten empfangen und lesen. Dabei ist es möglich Nachrichten einzeln und in größeren Gruppen zu empfangen. Wurde eine Nachricht einmal empfangen, wird sie unwiederbringlich gelöscht.

## Aufgaben

1. Aufgaben anlegen

Aufgaben sind immer einem Projekt zugeordnet. Aufgaben können daher nur angelegt werden, wenn ein Benutzer mindestens einem Projekt angehört. Im Stan­dard­fall wird eine neue Aufgabe zu dem Backlog eines Pro­jekts hinzugefügt. Wenn bereits ein Kanban Board erstellt wurde, kann eine Aufgabe auch direkt dem Kanban Board hinzugefügt wer­den. Dort wird die Aufgabe in der Spalte „Todo“ abgelegt. In diesem Fall wird die Auf­ga­be nicht mehr Im Backlog angezeigt.

1. Benutzern Aufgaben zuweisen

Eine bestehende Aufgabe kann einem Mitglied des Projekt-Teams zur Bearbeitung zu­ge­wie­sen werden. Dabei ist es egal, ob die Aufgabe vorher bereits einer Person zugewiesen war oder nicht. Es ist auch möglich die Zuweisung aufzuheben und keine neue Zuweisung vor­zunehmen („NULL“). Der zu­gewiesene Benutzer wird über eine Nachricht informiert.

1. Aufgaben zu einem Kanban Board hinzufügen

Aufgaben können aus dem Backlog zu genau einem Kanban Board hinzugefügt werden. Dazu muss eines aus vielen möglichen Kanban Boards eines Projekts gewählt werden. Außerdem muss festgelegt werden, in welcher Spalte die Aufgabe abgelegt werden soll. Der Standardfall ist, dass die Aufgabe in der Spalte „Todo“ abgelegt wird.

1. Aufgaben bearbeiten

Die Attribute „Be­zeich­nung“, „Beschreibung“, „Komplexität“ und „zuständige Person“ (siehe oben) kön­nen nach der Erstellung einer Aufgabe jederzeit durch einen Benutzer verändert werden. Es bleibt zu beachten, dass die Bezeichnung der Aufgaben in einem Pro­jekt eindeutig ist.

1. Aufgaben löschen

Aufgaben können von allen Benutzern gelöscht werden. Dabei spielt es keine Rolle, ob die Aufgabe auf einem Kanban Board oder dem Backlog liegt.

## Benutzer

1. Benutzer-Konto registrieren

Um ein Benutzer-Konto zu registrieren, müssen ein Benutzername und ein Passwort bestimmt werden. Bei erfolgreicher Registrierung soll der Keycloak dem Benutzer direkt einen Access-Token zurückgeben. Bei einem Fehler wird ein passender Fehlercode zu­rückgegeben. Das Erstellen eines Eintrags in der Datenbank für das neue Be­nut­zer­kon­to wird von dem Keycloak erledigt und kann über die REST-API des Keycloaks getriggert wer­den. Für das Registrieren eines Benutzerkontos muss das Backend eine Anfrage le­dig­lich an den Keycloak weiterleiten.

1. Benutzer-Konto löschen

Wenn ein Benutzer sich entscheidet, sein Konto zu löschen, werden alle damit ver­bun­de­nen Daten unwiederbringlich gelöscht. Das Ergebnis des Löschvorgangs wird durch einen passenden HTTP-Status quittiert. Das Löschen des Be­nut­zer­kon­to-Datensatzes in der Da­ten­bank wird von dem Keycloak erledigt und kann über die REST-API des Keycloaks ge­trig­gert wer­den. Für das Löschen eines Benutzerkontos muss das Backend eine Anfrage le­­dig­lich an den Keycloak weiterleiten.

## Tags

1. Tag erstellen

Tags können mit einer Bezeichnung und einer Beschreibung erzeugt werden und sind einem Benutzer zugeordnet. Die Bezeichnung ist global eindeutig.

1. Tag bearbeiten

Die Beschreibung eines Tags kann im Nachhinein vom Erzeuger des Tags geändert wer­den.

1. Tag löschen

Ein Tag kann von seinem Erzeuger gelöscht werden. Dadurch werden automatisch alle Re­ferenzen von Auf­ga­ben entfernt.

1. Tags zuweisen

Aufgaben können über Tags mit Metadaten wie z. B. Versionsnummern uvm. versehen wer­den. Dabei ist der Zugriff nicht auf die eigens erzeugten Tags beschränkt.

# Entwurfsmuster

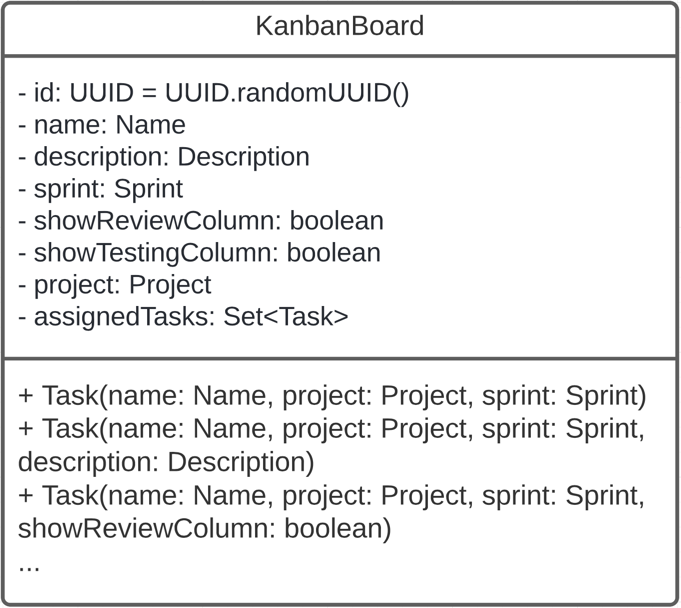
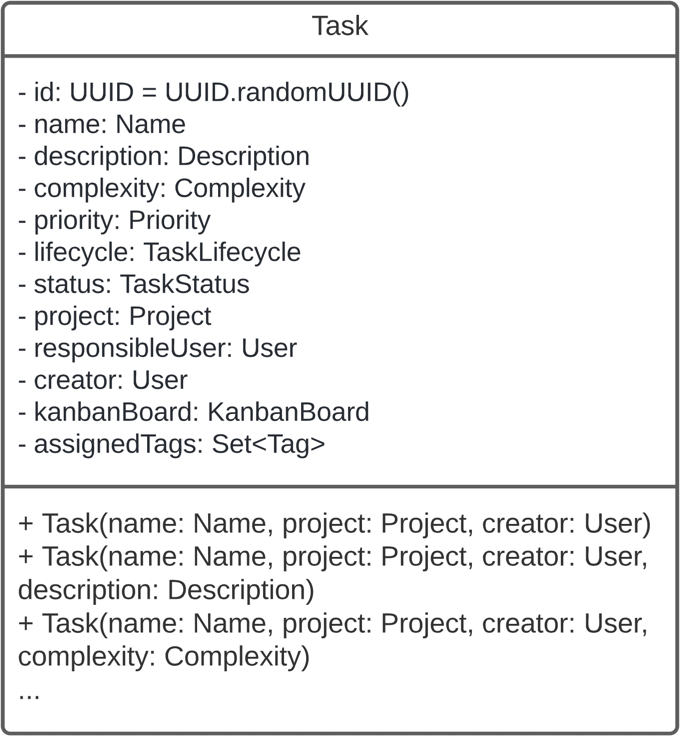
Das Verwenden von Entwurfsmustern ist eine Programmiertechnik, um häufig auf­tre­ten­de Probleme zu lösen. Entwurfsmuster sind nicht auf eine Problemdomäne be­­schränkt, son­dern beziehen sich auf allgemeine Problemstellungen. Nachfolgend wir­d bei­spiel­­haft ein in der Applikation Taskitory verwendetes Entwurfsmuster be­schrie­ben.

Das Erbauer Muster gehört zur Untergruppe der Erzeugungsmuster. Er­­­zeu­gungs­­muster wer­den verwendet, um die Instanziierung von Objekten zu regeln. Da­für wird die Logik ex­trahiert, die sich normalerweise in einem Konstruktor befindet. Mit Er­zeu­gungs­mus­tern wird festgelegt, welche Attribut-Belegungen erlaubt sind.

Mit einem Erbauer können die Attributwerte von Objekten schrittweise konfiguriert wer­den ohne, dass Objekte in einen unerlaubten Zustand initialisiert werden können. Dafür wer­den die Attribute einer Klasse in obligatorische und optionale Attribute unterteilt. Die op­tionalen Attribute werden mit Standardwerten belegt, sodass diese nicht explizit initia­li­siert werden müssen. Allerdings können diese Standardwerte explizit über­schrie­ben werden. Jedes optionale Attribut kann mit einer eigenen Methode überschrieben werden und ermöglicht den direkt anschließenden Aufruf äquivalenter Methoden.

In der Applikation Taskitory wird das Erbauer Muster für das Erzeugen von Objekten der Klas­­sen „Task“ (Aufgabe) und „KanbanBoard“ auf der Domänenschicht verwendet. Das Er­bauer Mus­ter bietet sich hier an, da Objekte dieser beiden Klassen eine Reihe von op­tio­nalen Pa­rametern besitzen. Die Programmiersprache Java unterstützt optionale Me­tho­den-Pa­ra­meter nicht in der Form, wie es z. B. Python tut. Daher müsste in diesen bei­den Fällen ei­ne große Menge von Konstruktoren definiert werden, die den Code un­über­sicht­lich und schlecht wartbar machen würde (siehe Abbildung 2).

Das Erbauer Muster wird für Aufgaben durch die Datei „CreateTask.java“ umgesetzt. Die Ent­­haltene Klasse „CreateTask“ definiert die gleichen Attribute wie eine Aufgabe mit Aus­nah­me der ID. Um den Ar­beitsprozess des Erbauers anzustoßen, muss die statische Me­tho­de „Create­Task.named(String name)“ aufgerufen werden. Dadurch wird die Klasse Cre­ateTask über den pri­vaten Konstruktor mit Standard-Attributwerten und dem ob­li­ga­to­rischen Namen in­stan­ziiert. Um den Erbauungs-Prozess fortzusetzen, muss die Me­tho­de „forProjectWithCreator(Project project, User creator)“ auf dem erzeugten Objekt auf­ge­rufen werden, mit der die obligatorischen Attribute „Projekt“ und „Autor“ festgelegt wer­­den. Der Rück­gabetyp dieser Methode ist das Interface „Builder<KanbanBoard>“.



Ca. 8! Konstruktoren

Ca. 4! Konstruktoren

Abb. 2: Klassen „Task“ und „KanbanBoard“ ohne Erbauer Muster

Die Klasse „CreatableKanbanBoard“ implementiert dieses Interface. Die­ Klasse ist unter der Klasse „CreateTask“ definiert und wird verwendet, um die op­tio­­nalen Parameter ei­ner Aufgabe bei Bedarf festzulegen. Dazu wird für jeden optionalen Pa­­rameter eine Me­tho­de definiert. Diese Methoden haben alle Rückgabewerte vom Typ „Crea­tableTask“, so­dass die Methoden kontinuierlich aufgerufen werden können. Um den Er­­bauungs-Prozess ab­zuschließen, muss die Methode „build()“ aufgerufen werden, die mit dem Konstruktor der Klasse „Task“ eine Objekt erzeugt. Um zu verhindern, dass Auf­ga­ben auf anderem Weg er­zeugt werden, wird die Sichtbarkeit der Konstruktoren der Klas­se „Task“ ein­ge­schränkt. Durch die Verwendung des Erbauer Musters verändert sich die Klassen-Struk­tur aus Ab­bil­dung 2 zu der Klassenstruktur in Abbildung 3.

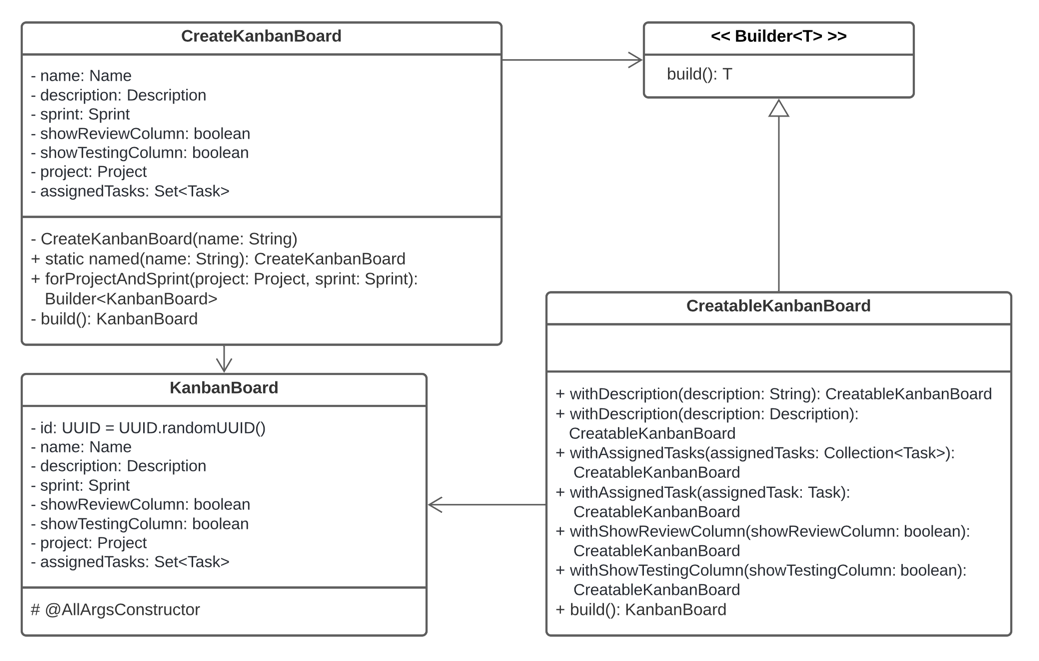


Abb. 3: Die Kanbanboard Entität mit integriertem Erbauer Muster

# Programmierprinzipien

Programmierprinzipien sind Richtlinien für das Schreiben von verständlichem und ein­fach verwendbarem Quellcode. Es gibt eine Reihe von Programmierprinzipien, die in ver­schie­denen Kombinationen und mit verschiedenen Definitionen auftreten. Zwei weit ver­brei­tete Listen von Programmierprinzipien sind die SOLID- und die General responsibility As­signment Patterns. Nachfolgend wird auf die in diesen Listen enthaltenen Pro­gram­mier­prinzipien und das DRY-Prinzip eingegangen. Es wird außerdem beschrieben, wie und wo diese Programmierprinzipien in der App­li­kation Taskitory verwendet wurden.

## SOLID

### Single Responsibility Principle

Das Single Responsibility Principle besagt, dass Quellcode-Einheiten nur eine Aufgabe er­fül­­len sollen. Das Prinzip ist auf Mo­du­le, Klas­sen, Methoden und Variablen anwendbar.

In der Applikation Taskitory wurde das Single Responsibility Principle regelmäßig auf der Do­­mänenschicht verwendet. Die Value Objects sind dazu gedacht inhaltlich zu­sam­men­ge­­hörige Funktionalität zu kapseln und dadurch aus den Entitäten zu extrahieren. Ein Bei­spiel dafür ist das Value Object „ProjectSecurityKey“. In dieser Klasse werden alle Funk­tio­nen für die Verwaltung von den Projekt Sicherheitsschlüsseln implementiert. Es wer­den außerdem die Zugriffsmöglichkeiten auf Projekt Sicherheitsschlüsseln definiert.

### Open Closed Principle

Das Open Closed Principle besagt, dass Klassen und Module offen für Erweiterungen, aber ver­schlossen für Modifikationen ist. Klassen sollen nicht durch Veränderung der Klasse selbst, sondern durch Vererbung um typspezifische Funktionen erweitert werden.

Das Open Closed Principle wird durch die Nachrichten Entität in der Applikation Tas­ki­tory umgesetzt. Die allgemeine Struktur von Nachrichten wird durch eine abstrakte Va­ter­klasse definiert. Die Konkreten Implementierungen dienen als Spezialisierungdieser all­gemeinen Struktur. Eine Erweiterung der Anzahl verschiedener Nachrichten ist je­der­zeit durch Vererbung möglich. Die grundlegende Struktur von Nachrichten ist allerdings nicht veränderbar.

### Liskov Substitution Principle

Das Liskov Substitution Principle besagt, dass Sohnklassen an jeder Stelle im Quellcode an­stelle einer Vaterklasse verwendet werden kann ohne, dass ein Fehler auftritt.

Es kann mehrere verschiedene Arten von Nachrichten in der Applikation Taskitory geben. Al­le diese Nachrichten werden durch Sohnklassen einer übergeordneten abstrakten Va­ter­klasse umgesetzt. Um die verschiedenen Nachrichten persistieren zu können, wird der Nach­richten-Typ zusätzlich durch ein Attribut repräsentiert. Beim aktuellen Stand sind kei­ne speziellen Funktionen der Sohnklassen implementiert, sodass ein Austausch der Va­ter- mit der Sohnklasse keinen Unterschied machen würde.

### Interface Segregation Principle

Das Interface Segregation Principle besagt, dass Interfaces aufgabenspezifisch sein soll­ten. Es sollen möglichst keine zu allgemeinen Interfaces verwendet werden.

Die Repositories auf der Domänen Schicht sind nur für ein bestimmtes Aggregat zu­stän­dig. Dadurch ist der Aufgaben Bereich klar fokussiert und abgegrenzt. Durch eine Kom­po­si­tion der Repositories (siehe Services auf der Applikations Schicht) können alle Funk­tio­nen verwendet werden. Allerdings können auch gegebenenfalls durch Weglassen eines Re­positories die Abhängigkeiten reduziert werden.

### Dependency Inversion Principle

Das Dependency Inversion Principle besagt, dass Abstraktionen nicht von technischen De­tails abhängen dürfen. Umgekehrt sollen Details von Abstraktionen abhängen.

Die Applikation Taskitory hat eine Clean Architecture. Dadurch sind technische Im­ple­men­tierungen von der allgemeinen Domänenlogik abhängig. Da fällt besonders bei Be­trach­tung dem OR-Mapping auf. Dabei werden auf der Domänenschicht die allgemeinen Schnitt­stellen (Repositories) definiert, über die Objekte einer Entität angefragt werden kön­nen. Auf der Pluginschicht befindet sich die technische Implementierung dieser Schnitt­stellen (SpringDataRepositories), die das Interface „JpaRepository“ erweitern. Auf der allgemeineren Schicht werden somit die Anforderungen an eine technische Im­ple­men­tierung definiert. Es besteht keine Abhängigkeit von diesen technischen Im­ple­men­tie­rungen. Umgekehrt hängen die technischen Implementierungen von der Schnitt­stel­len­definition ab, da sie die definierten Anforderungen erfüllen müssen.

## GRASP

### Geringe Kopplung

Unter „Kopplung“ wird der Grad der Abhängigkeit zwischen Objekten verstanden. Die Ab­hän­gigkeiten zwischen Quellcode mit verschiedenen Aufgaben sollten minimiert und so lo­se wie möglich gehalten werden.

Die Architektur der Applikation Taskitory sind so konzipiert, dass die inneren Schichten von keiner weiter außen liegenden Schicht abhängen. Eine besonders Lose Kopplung wird durch die Repositories auf der Domänen Schicht umgesetzt. Denn diese Interfaces de­fi­nie­ren nur eine Schnittstelle, von der die konkreten Implementierungen auf der Plugin Schicht abhängen.

### Hohe Kohäsion

Unter „Kohäsion“ wird die räumliche Nähe von inhaltlich zusammengehörigem Quellcode be­zeichnet. Es sollte das Ziel sein, inhaltlich zusammengehörigen Quellcode möglichst na­he zusammenzuhalten.

Eine Hohe Kohäsion besteht auf der Domänenschicht besonders durch die verwendeten Va­lue Objects. Wie bereits beim Single Responsibility Principle beschrieben wurde, kap­seln diese Value Objects ganz bestimmtes inhaltlich zusammengehöriges Verhalten. Auch die Entitäten grenzen Quellcode inhaltlich ab. Eine Ausnahme dazu sind die Aggregate Roots, die als einziger zentraler Zugriffspunkt auf die Entitäten eines Aggregats auch Me­tho­den für den Zugriff auf die Funktionen dieser enthaltenen Entitäten ermöglichen.

### Information Expert

Als „Information Expert“ wird eine Klasse bezeichnet, die allen Quellcode für eine be­stimm­te Aufgabe enthält. Der gesamte diese Aufgabe betreffende Quellcode soll in dieser ei­nen Klasse gekapselt werden. Interne Informationen dürfen die Klasse nicht verlassen.

Wie bereits beim Single Responsibility Principle beschrieben wurde, kapseln diese Value Objects ganz bestimmtes inhaltlich zusammengehöriges Verhalten. Zum Beispiel wird in der Klasse „Sprint“ die Validierung von Sprintzeiträumen implementiert. Zusätzlich kann geprüft werden, ob ein Sprintzeitraum bereits abgelaufen ist. Alle einen Sprintzeitraum be­tref­fen­den, für die Applikation Taskitory relevanten Funktionen sind und werden zukünftig aus­schließlich in dieser Klas­se abgelegt. Das Geheimnisprinzip lässt sich be­son­ders gut an dem Value Object „ProjectSecurityKey“ erkennen. Denn der eigentliche Si­cher­heits­schlüssel kann nicht im Klartext aus dem Objekt extrahiert werden. Lediglich ein kryp­tographischer Hash-Wert dieses Sicherheitsschlüssels kann herausgegeben werden.

### Polymorphie

Unter „Polymorphie“ wird das unterschiedliche Implementieren von Verhalten durch Über­schreiben von Methoden in einer Vererbungsbeziehung verstanden.

Die Nachrichten Entität wurde durch ein Refactoring (siehe Kapitel 7.1) so angepasst, dass konkrete Nachrichten von einer allgemeinen, abstrakten Klasse abgeleitet werden. Da­durch wird Polymorphie theoretisch implementierbar. Allerdings wird bei dem ak­tu­el­len Stand keine Funktion für Nachrichten implementiert, die Polymorphie bedarf. Denk­bar wäre eine Verschlüsselung des Inhalts bei bestimmten Nachrichten-Typen.

### Pure Fabrication

Das Pure Fabrication Prinzip besagt, dass es Klassen in der Domäne geben darf, die Funktionen zur Verfügung stellen, für die sie kein Information Expert sind.

Der einzige definierte Domain Service der Applikation Taskitory ist eine Klasse mit einer Rei­he von statischen Methoden, die das Verhältnis zwischen einem Benutzer und einer Pro­jekt-Ressource überprüfen. Diese Klasse ist kein Information Expert für dieses Ver­hält­nis. Das wird besonders durch die Projekt Mitgliedschafts Entität deutlich, die das Ver­hältnis eines Benutzers zu einem Projekt und damit auch seinen Ressourcen definiert. In dieser Entität werden auch Funktionalitäten definiert, die das Verhältnis zwischen Be­nut­zer und Projekt-Ressource beeinflusst.

### Delegieren

Verhalten sollte sofern möglich an Information Experts delegiert werden.

Entitäten lagern inhaltlich zusammengehöriges Verhalten an Value Objects aus. Zum Bei­spiel wird die Validierung des Sprintzeitraums von der Kanban Board Entität an das Sprint Value Object delegiert.

### Controller

Ein Controller ist die Schnittstelle zwischen dem Frontend und dem Backend. Controller lei­ten Systemereignisse an die richtigen Stellen weiter.

Die Applikations und die Adapter Schicht dienen als Schnittstellen für die Kommunikation der Anwendungslogik auf der Domänen Schicht und der Benutzerschnittstelle auf der Plug­in Schicht. Die relevanten Informationen der auf der Plugin Schicht eingehenden HTTP Anfragen werden von der Adapterschicht umgeformt, damit sie von der App­li­ka­tions Schicht verarbeitet werden können. Die Applikatins Schicht zerlegt die Anfrage in ih­re Bestandteile und leitet diese an die zuständigen Stellen in der Domänen Logik weiter.

### Creator Principle

Das Creator Principle definiert, wer Instanzen von einer Klasse erzeugen darf. Eine Klasse A darf Instanzen einer Klasse B erzeugen, wenn A Objekte von B enthält, diese verarbeitet oder von diesen abhängt. Es ist außerdem erlaubt, wenn A der Information Expert für das Er­zeugen von B-Instanzen ist.

Das Creator Principle wird durch die Aggregate Roots auf der Domänenschicht und die Builder (siehe Kapitel 5) umgesetzt. Die Aggregate Roots sind die zentralen Verwalter der in einem Aggregat enthaltenen Entitäten und müssen daher Instanzen der enthaltenen En­titäten erzeugen können. Die Builder-Klassen sind Information Experts für das Er­zeu­gen von Instanzen der jeweiligen Klassen (Kanban Board und Aufgaben Entitäten) und müs­sen daher ebenso Instanzen dieser Klassen erzeugen können.

## DRY

DRY steht für Don’t Repeat Yourself. Dieses Prinzip besagt, dass keine Redundanzen im Quell­­code vorkommen sollen.

Das DRY Prinzip wird in der Applikation wesentlich dadurch umgesetzt, dass das OR-Map­ping auf der Domänen Schicht umgesetzt wird. Theoretisch müsste sich das OR-Map­ping als technische Implementierung auf der Pluginschicht befinden. Allerdings müss­te dafür der Domänen-Code zu einem Großteil dupliziert werden. Zusätzlich müsste ein Mapping zwischen den beiden Schichten aufgebaut werden. Diese Dopplung hätte zu enor­mem unnötigem Aufwand bei der Entwicklung der Applikation Taskitory geführt. Da­­her wurde das OR-Mapping auf der Domänen Schicht umgesetzt.

# Code Smells und Refactoring

Code Smells sind Strukturen im Quellcode, die die Entwicklung, Wartung und/oder das Tes­ten behindern. Um diese Strukturen zu beseitigen oder zu verbessern, wird ein Re­fac­to­ring durchgeführt. Dabei werden dysfunktionale Strukturen identifiziert und durch funk­tionale Strukturen ersetzt. Nachfolgend werden zwei Code Smells in dem Quellcode der Applikation Taskitory identifiziert und durch ein Refactoring beseitigt.

## Open Closed Principle Verstoß

Die Nachrichten Entität auf der Domänen Schicht unterscheidet mit dem Attribut „type“ zwi­schen verschiedenen Arten von Nachrichten. Um unterschiedliches Verhalten der glei­chen Methode bei unterschiedlichen Nachrichten zu implementieren, müssten vor dem Re­factoring If- oder Switch-Anweisungen über das besagte Attribut verwendet werden. Das entspricht nicht dem Open Closed Principle (siehe Kapitel 6) und kann daher als Code Smell angesehen werden. Erweiterungen oder Änderungen würden aufwändige An­passungen in der Klasse erfordern. Zudem werden solche Unterscheidungen bei ei­ner großen Anzahl von verschiedenen Nachrichts-Typen schnell unübersichtlich. Abbildung 4 stellt den Zustand vor dem Refactoring dar.

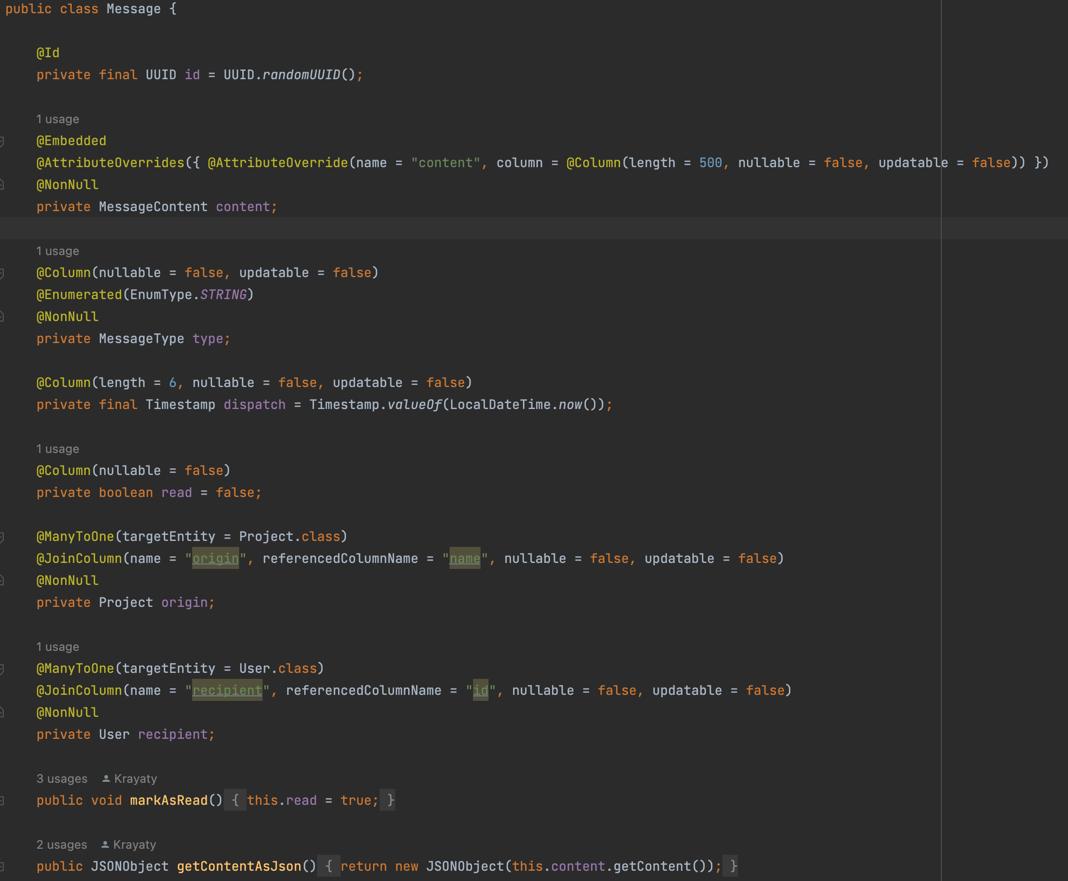
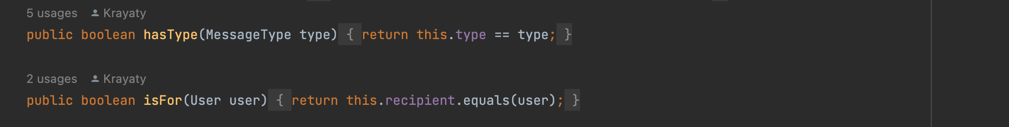


Abb. 4: Die Nachrichten Entität

Um das Open Closed Principle für die Nachrichten Entität auf der Domänen Schicht um­zu­setzen, werden extra Klassen für die verschiedenen Nachrichten erstellt und von der all­gemeinen Nachrichten-Klasse abgeleitet. Um nur valide Nachrichten erstellen zu kön­nen, wird eine Factory verwendet und die Sichtbarkeit des Konstruktors der Nachrichten Va­­terklasse eingeschränkt. Die Nachrichten Vaterklasse kann zusätzlich als ab­strakt de­fi­niert werden. Die Abbildungen 5 und 6 zeigen den Zustand nach dem Refactoring.

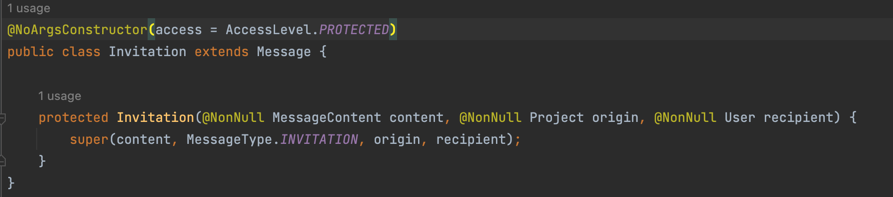
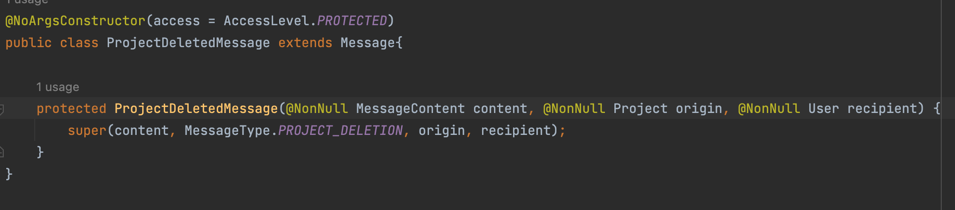


Abb. 5: Die konkreten Nachrichten Implementierungen

Beim jetzigen Stand der Applikation Taskitory sind zwar noch keine polymorphen Me­tho­den enthalten. Allerdings ist eine Implementierung von polymorphem Verhalten durch die­ses Refactoring deutlich einfacher und übersichtlicher machbar.

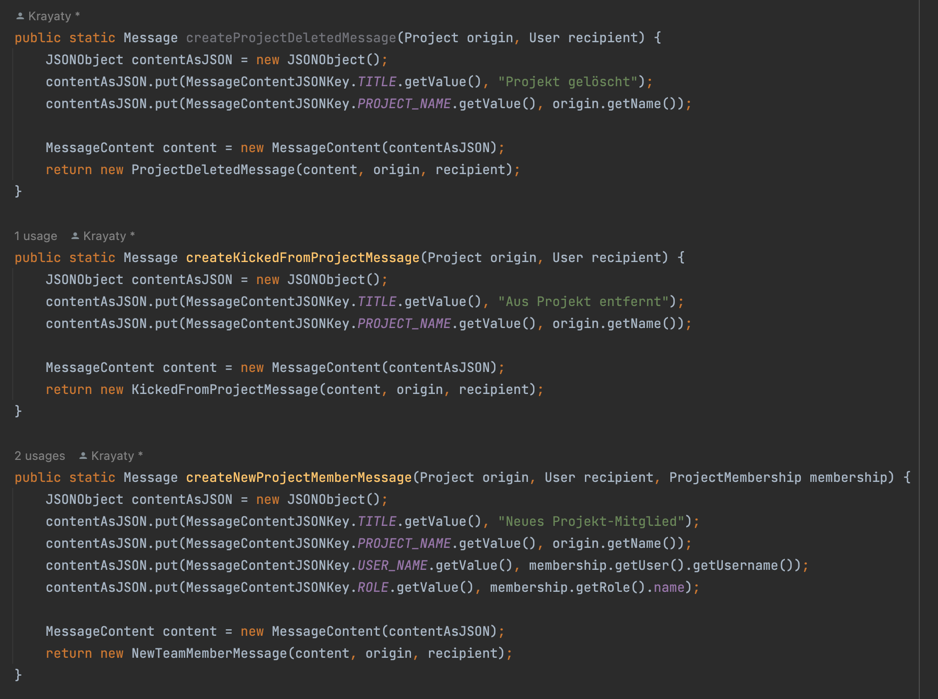


Abb. 6: Die Factory-Methoden für das erzeugen konkreter Nachrichten

## DRY Prizip Verstoß

Das Value Object „Sprint“ auf der Domänen Schicht kapselt hauptsächlich die Erzeugung und Validierung von Sprintzeiträumen. Dabei kommen mehrere Konstruktoren mit un­ter­schiedlichen Parametern zum Einsatz. Dem entsprechend müssen die gleichen Va­li­die­rungs-Regeln in verschiedenen Konstruktoren angewendet werden. Vor einem Re­fac­to­ring wurde die Validierungs-Logik in jedem Konstruktor redundant implementiert. Das ver­stößt klar gegen das DRY Prinzip (siehe Kapitel 6) und kann daher als Code Smell be­wer­tet werden. Denn durch diese Dopplungen können Inkonsistenzen in der Va­li­die­rungs­logik auftreten und der Wartungsaufwand ist relativ hoch. Die Abbildungen 7 und 8 zei­gen den Zustand vor einem Refactoring.

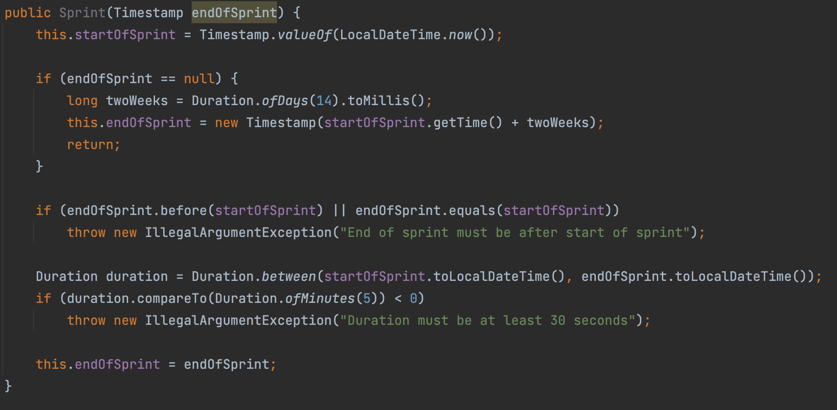


Abb. 7: Die Konstruktoren des Value Objects „Sprint“ (Teil 1)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abb. 8: Die Konstruktoren des Value Objects „Sprint“ (Teil 2)

Durch Extrahieren der Validierungslogik für Sprint-Zeiträume und Sprint-Endzeitpunkte in eine eigene statische Methode, wird die Redundanz beseitigt. Bei einer Änderung der Va­­lidierungs-Logik muss nun nur noch die jeweilige statische Methode verändert werden und nicht mehr jeder betroffene Konstruktor. Die Abbildungen 9, 10 und 11 zeigen den Zu­­­stand nach dem Refactoring.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abb. 9: Die refactorierten Sprint-Konstruktoren (Teil 1)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abb. 10: Die refactorierten Sprint-Konstruktoren (Teil 2)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abb. 11: Die extrahierte Validierungslogik für Sprints