TASKCHAIN

Dario Schaffner

Dokumentation M-223
Multi-User-Applikationen objektorientiert realisieren
Abgabe: 24.03.2022
Fachexperte: Steinmann Remo

Inhalt

1 Vorwort

Diese Dokumentation gehört zum Modul-223 und wurde von Dario Schaffner im Rahmen der Entwicklung der Taskchain Applikation erstellt. Sie dient dazu, das Vorgehen und den Ablauf der Arbeit zu dokumentieren und alle relevanten Informationen zu erfassen, die während des Entwicklungsprozesses anfielen. Die Dokumentation gliedert sich in zwei Teile, die jeweils unterschiedliche Aspekte der Arbeit abdecken.

Der erste Teil der Dokumentation stellt die ausgearbeitete Aufgabenstellung vor und zeigt den Ablauf der Arbeit auf. Hierbei wird auch erläutert, mit welchen Mitteln gearbeitet wurde und welche Vorkenntnisse der Schüler bereits besass. Hierbei werden auch die Methoden und Werkzeuge aufgezeigt, die im Rahmen des Projekts zum Einsatz kamen.

Im zweiten Teil der Dokumentation wird die eigentliche Arbeit dokumentiert, einschliesslich der aufgetretenen Hindernisse und dem Testen der Applikation. Hier wird auch beschrieben, wie die Projektplanung von Anfang bis Ende nach der IPERKA-Methode erfolgte und wie der Schüler mit unerwarteten Schwierigkeiten umging.

Die Dokumentation endet mit einer Auswertungsphase, in der eine Reflexion und ein Fazit präsentiert werden. Hier wird die Arbeit kritisch reflektiert und bewertet, um Verbesserungen und Optimierungspotenziale zu identifizieren. Die Reflexion und das Fazit bilden somit einen wichtigen Bestandteil der Dokumentation und geben einen Einblick in die Erfahrungen und Erkenntnisse des Schülers bei der Entwicklung der Taskchain Applikation.

2 Aufbau und Ablauf (Teil 1)

2.1 Aufgabenstellung

Die Projektarbeit des Moduls 223 stellt eine herausfordernde Aufgabe für den Schüler dar, da er nicht nur eine Software erstellen muss, sondern auch sicherstellen muss, dass sie Objektorientiert implementiert wird und eine Datenbankintegration enthält. Dies erfordert ein tiefes Verständnis der Programmierung, insbesondere der objektorientierten Konzepte, sowie der Datenbanktechnologie.

Die Freiheit bei der Wahl der Technologie und der detaillierten Anforderungen an die Applikation gibt dem Schüler die Möglichkeit, seine Fähigkeiten und sein Wissen in einer Umgebung anzuwenden, die der realen Welt ähnlich ist. Indem er die Technologie und die Anforderungen selbst definiert, kann er auch seine Präferenzen und Interessen in seinem Projekt zum Ausdruck bringen.

Um ein erfolgreiches Projekt zu erstellen, muss der Schüler sorgfältig planen, entwerfen, implementieren, testen und dokumentieren. Er muss auch sicherstellen, dass alle Aspekte des Projekts abgedeckt sind und dass alles innerhalb des vorgegebenen Zeitrahmens abgeschlossen wird.

2.1.1 Ausgangslage

Die Projektarbeit im Modul 223 wird dafür genutzt, die Prozesse, Formalitäten, Beurteilungsmodalitäten und Rahmenbedingungen für die Abschluss IPA im schulischen Kontext zu erfahren bzw. zu üben. Sie sollen durch eine möglichst praxisnahe Umsetzung einen ersten Einblick in die Durchführung dieser Probe IPA erhalten und daraus, neben den fachlichen Inhalten, auch wichtige Erkenntnisse hinsichtlich der Planung und Durchführung der IPA erhalten.

2.1.2 Aufgabenstellung im Projektbeschreib

Fachthema
Eine objektorientierte Multi-User-Applikation wird geplant, umgesetzt, getestet und dokumentiert. Die objektorientierte Applikation erfüllt folgende Kriterien:

- Front- und Backend
- Zentrale Datenbank
- Mehrere User greifen gleichzeitig auf den gleichen Datenbestand zu
- Zentrale Benutzer- und Rechteverwaltung

Aufgabenstellung

- Erstellt in der Freizeit eine Aufgabenstellung mit einem Umfang von 2-4 A4-Seiten. Dies gilt nicht als Arbeitszeit.
- Das Projekt muss komplett neu sein und darf keine Erweiterung eines bestehenden Projekts sein.
- Ein Projekt von der Abteilung und das Arbeiten auf Firmen-Infrastruktur (Notebook, Entwicklungsumgebung, Server) ist möglich, aber nicht obligatorisch (bei der Nutzung von Firmeninfrastruktur, die Erlaubnis von der Abteilung einholen).
- Die Wahl der Technologie ist euch überlassen (Rahmenbedingungen oben müssen aber immer eingehalten werden, => Empfehlung: Technologie von richtiger IPA verwenden).
- Beispiel für eine solche Aufgabenstellung findet ihr auf der Webseite der PK19 in Zürich www.pk19.ch (https://pk19.ch/wp-content/uploads/2020/12/Aufgabenstellung-API-Beispiel1.pdf)
- Das gesamte Projektresultat (Programmcode, Dokumentation und Präsentation) wird am Schluss abgegeben, bei uns archiviert sowie steht der Berufsbildung als Muster für zukünftige Durchführungen komplett zur Verfügung. Bitte geeignete Projekte wählen.

Termine und Zeiten

- Das praktische Projekt dauert 6 Tage (Kurstage) und die täglichen Arbeitszeiten sind von 08:15h-11:30h und 12:30h-16:15h (7h/Tag). Der Abgabetermin des Projektes ist am 6. und letzten Tag um 11:30h (Abgabe Dokumentation).
- Der Umfang des Projektes sind 5x6h plus 3h am Abgabetag (Total 33h). Dazu kommt in den ersten fünf 1h Theorie/Tag und die Abschlussarbeiten am Nachmittag des sechsten Tages von rund 4h.
- Als Vorbereitung bzw. Nachbearbeitung kann mit 6h für die Erstellung der Aufgabenstellung, der Präsentation und der Demonstration gerechnet werden.
- Täglich soll rund 60% an der Entwicklung und Umsetzung des Projektes und rund 40% an der Dokumentation des Projektes gearbeitet werden.
- Die Vorbereitung/Erstellung der Aufgabenstellung, der Präsentation und der Demonstration werden in der Freizeit realisiert und zählen nicht zur Arbeitszeit.

Detaillierte Aufgabenstellung

Die Aufgabe umfasst das Erstellen einer dynamischen Webseite mit einem Backend und einer Datenbank welche dazu dienen sollen die nötigen Daten zu speichern und zu verarbeiten. Es soll funktional möglich sein sich als Benutzer zu registrieren, anzumelden, Tickets zu erfassen sowie auch zugeteilte Tickets zu verwalten. Bei dem Projektantrag wurden noch 3 individuelle Kriterien definiert.

Akzeptanzkriterien

- Realisierung der Applikation
- Saubere Dokumentation
- Pünktliche Abgabe gemäss definiertem Datum

Nicht-Funktionale Anforderungen

- Der Code-Style im Frontend soll formatiert, lesbar und sauber kommentiert sein
- Der Code-Style im Backend soll formatiert, lesbar und sauber kommentiert sein
- Die Passwörter der Benutzer sollten sicher gespeichert werden

Funktionale Anforderungen

- Benutzer können sich mit einem Benutzernamen und Passwort registrieren und einloggen
- Jeder Benutzer kann ein «Board» erstellen, was ihn zu einem Admin dieses Boards macht.
- Administratoren können Benutzer innerhalb eines Boards verwalten
- Tickets können mit Checklisten erfasst werden
- Tickets können einem Benutzer zugewiesen werden
- Tickets, in denen die Checkliste abgeschlossen wurde, werden grün markiert.
- Tickets können in verschiedene Spalten verschoben werden
- Tickets können gelöscht werden
- Der zugewiesene Benutzer kann die Checkliste Bearbeiten
- Der zugewiesene Benutzer kann das Ticket verlassen
- Der zugewiesene Benutzer kann URLs zu aktuellen Test-Runs / Pullrequests hinzufügen.
- Alle Benutzer des Boards können die aktuellen Tickets sehen

2.1.3 Individuelle Bewertungskriterien

Drei individuellen Kriterien aus Kriterienkatalog, ohne Mehrfachbewertungen, wählen (in der effektiven IPA werden sieben individuelle Kriterien gewählt).

1. Individuelles Bewertungskriterium

Nummer Katalog-Kriterium - Bezeichnung	
194 - Plausibilisierung der Benutzer-Eingaben	
Definition (Leitfrage)	
Werden die Eingaben des Benutzers überprüft	
Gütestufe 3	*Gütestufe 2*
Alle Eingabefelder werden überprüft. Es Ist eindeutig gekennzeichnet, welche Felder Pflichtfelder sind. Für den Benutzer ist ersichtlich, welche Wertebereiche zulässig sind. Findet die Plausibilisierung eine Fehleingabe, so wird der Benutzer mit konkreten Hinweisen geführt, das entsprechende Feld wird aktiviert.	Plausibilisierung findet statt, Feedback an Benutzer ist mangelhaft/nicht eindeutig/unvollständig. Nur korrekte Daten werden übermittelt.
Gütestufe 1	*Gütestufe 0*
Eingaben werden plausibilisiert, aber bei Fehlern oder fehlenden Eingaben sind die bisher gemachten Eingaben verloren oder die fehlerhaften Eingaben werden trotzdem übermittelt. Oder: es werden nicht alle Eingaben überprüft, welche überprüft werden sollten.	Es findet keine Plausibilisierung statt.

2. Individuelles Bewertungskriterium

Nummer Katalog-Kriterium - Bezeichnung GUI-Design (Fokus: Benutzerfreundlichkeit über die ganze Applikation)	
Definition (Leitfrage) Ist die Applikation als Ganzes benutzerfreundlich? 1. Masken in der richtigen Reihenfolge (Applikation bildet den Prozess/Workflow richtig ab). 2. Ist das Design durchgängig (gleiche Elemente am gleichen Platz) 3. Kann der Benutzer bei Fehleingaben zurück navigieren (oder ist für den Benutzer transparent dargestellt, warum dies nicht möglich ist)?	
Gütestufe 3 Alle 3 Punkte sind erfüllt.	*Gütestufe 2* Alle 2 Punkte sind erfüllt.
Gütestufe 1 1 Punkt erfüllt.	*Gütestufe 0* Kein Punkte ist erfüllt.

3. Individuelles Bewertungskriterium

Nummer Katalog-Kriterium - Bezeichnung Kommentare im Quellcode	
Definition (Leitfrage) Wurde der Sourcecode der Applikation ausreichend kommentiert?	
Gütestufe 3 Der Sourcecode der Applikation ist vollumfänglich kommentiert: 1. Funktionen, Parameter, Rückgabewerte, 2. Wichtige Stellen im Sourcecode, 3. weitere zusätzliche/nützliche Kommentare.	*Gütestufe 2* Der Sourcecode der Applikation ist im Grossen und Ganzen kommentiert. Einer der genannten Punkte könnte präziser sein.
Gütestufe 1 Der Sourcecode der Applikation ist nur teilweise kommentiert.	*Gütestufe 0* Der Sourcecode der Applikation ist unzureichend kommentiert.

2.1.4 Technologien

- Angular
  - Front-End-Webapplikationsframework
  - Basiert auf Typescript, welches Objektorientiert ist
  - Cross-Plattform
  - Entwickelt von Google
- .NET Core 6.0 (C#)
  - Software-Plattform, für die Entwicklung und Ausführung von Anwendungsprogrammen
  - Objektorientiert
- MongoDB
  - No-SQL Datenbankmanagementsystem
  - Daten werden in JSON-ähnlichen Dokumenten verwaltet / gespeichert
  - Flexibel und skalierbar
  - Schnelle abfragen

2.2 Projektaufbauorganisation

2.2.1 Projektmitglieder

Am Projekt Taskchain sind folgende Personen beteiligt

Name	Rolle
Dario Schaffner	Projektmitglied, Entwickler
Remo Steinmann	Experte, Fachvorgesetzte Person

2.2.2 Projektmethode

Die IPERKA Methode ist eine sehr strukturierte und systematische Vorgehensweise bei der Durchführung von Projekten. Sie basiert auf der Analyse und Ausführung von Aufträgen Punkt für Punkt und ermöglicht es, dass der Schüler genau weiß, welche Schritte in welcher Reihenfolge durchgeführt werden müssen.

Durch die klare Trennung der Projektphasen ist es einfacher, den Fortschritt des Projekts zu überwachen und Probleme rechtzeitig zu identifizieren. Dadurch können Änderungen schnell und effektiv umgesetzt werden, um das Projekt wieder auf Kurs zu bringen.

Aufgrund dieser Stärken und den positiven Erfahrungen aus früheren Modulen hat der Schüler sich für die Verwendung der IPERKA Methode in diesem Projekt entschieden.

2.2.3 Git

Git ist ein verteiltes Versionskontrollsystem, das für die Verwaltung von Code-Repositories verwendet wird. Es ermöglicht Entwicklern, Änderungen an Code-Basis zu verfolgen und gemeinsam zu arbeiten, indem sie Änderungen in verschiedenen Zweigen und Forks verwalten.

Git wurde in dieser Projektarbeit für die Code Versionierung verwendet. Es wurde ein privates Repository auf GitHub erstellt namens «Taskchain». Dieses Repository wurde verwendet um den Code zu speichern. Jedes mal wenn eine Änderung gemacht wurde, erstellte der Schüler ein Commit welches dazu führt, dass eine Code Versionierung entsteht.

2.3 Zeitplan

2.3.1 Vorgegebene Termine

Datum	Termin
27.02.2023	Infotag
14.03.2023	Start der Probe IPA 1. Besuch des Experten / Besprechung des Zeitplans und der Dokumentation
22.03.2023	2. Besuch des Experten
24.03.2023	Abgabe Projektarbeit

2.4 Geplante Arbeiten

Auf der nächsten Seite finden sie der ausgefüllte Zeitplan.

Der Zeitplan ist auch noch online unter folgendem Link zu finden: https://i.imgur.com/c2OJubm.png

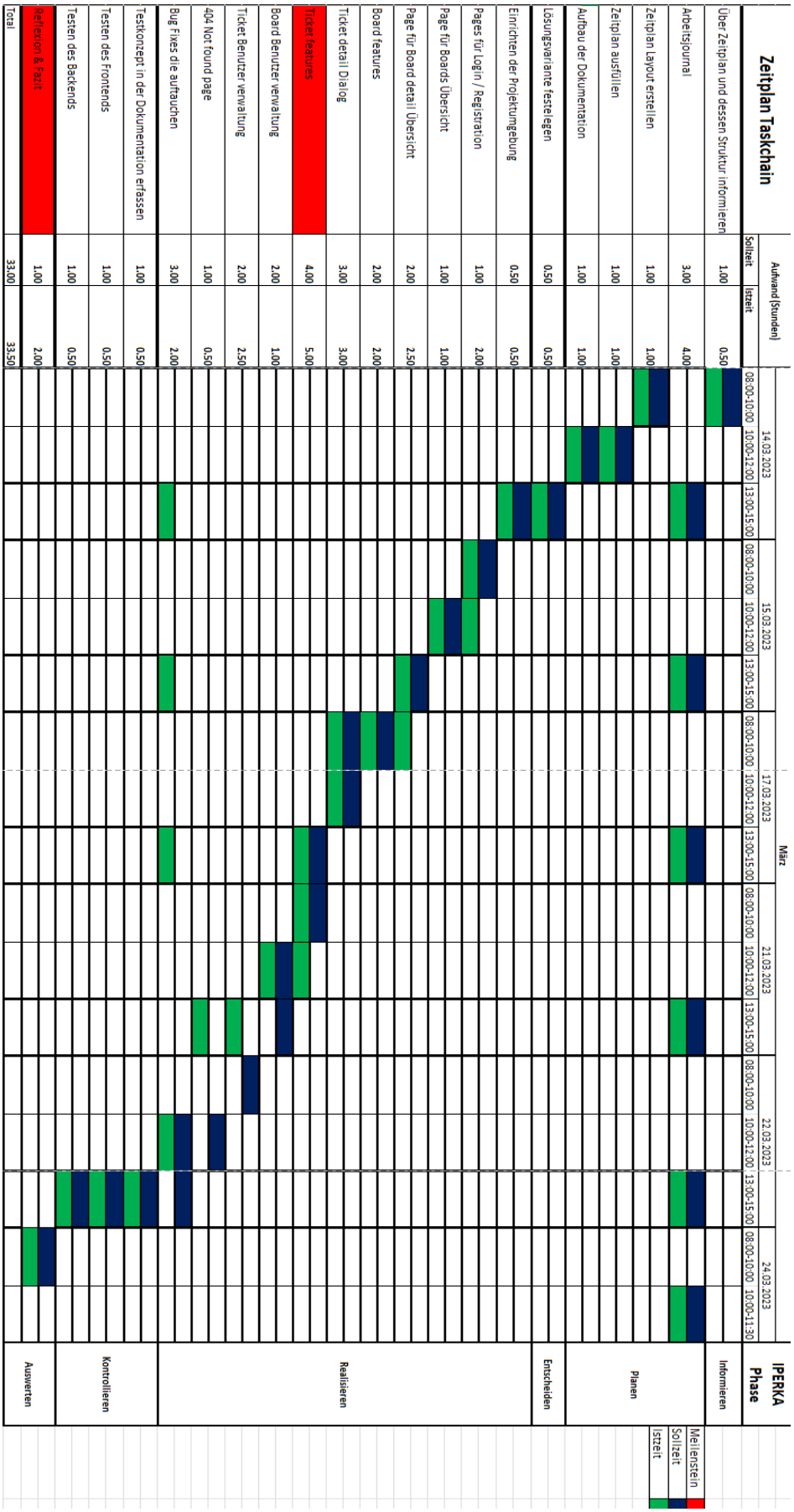

Abbildung 1 - Zeitplan

3 Arbeitsjournal

Projekttag 1 (14.03.2023)

	Dario
Geplante Aktivitäten	• Informieren über den Zeitplan • Zeitplan Layout erstellen • Dokumentation Grundstruktur erstellen • Einrichten der Projektumgebung
Durchgeführte Arbeit	• Zeitplan erstellt und ausgefüllt • Grundstruktur für die Dokumentation erstellt • Basis Struktur für das Front und backend erstellt, sowie ein Repository auf GitHub.
Erfolge & Misserfolge	Der Start der Probe IPA war etwas holprig, da wir externen eine Stunde lang in dem Empfangsbereich warten mussten, bis wir eine Badge mit Zugang zu dem Siemens Gebäude erhalten haben Diese Stunde hätten wir besser nutzen können, um den Zeitplan sowie auch andere Aufgaben zu erledigen. Das Erstellen des Zeitplans war zuerst eine eher schwierige Aufgabe, da ich nicht sicher war, wie dieser strukturiert werden soll. Nach dem Input von Lara, habe ich entschieden die verschiedenen Modultage in 2Std Blöcke aufzuteilen. Der Zeitplan, die Grundstruktur für die Dokumentation sowie die Basis Struktur wurde erfolgreich erstellt.
Hilfestellung	Der Zeitplan und die Grundstruktur der Dokumentation wurden mit dem Experten besprochen. Bei dem Zeitplan haben wir bemerkt, dass ich etwas zu genau und zu viel Zeit eingeplant haben. Wir haben uns darauf geeinigt, dass ich den Zeitplan noch einmal überarbeite, und schaue, ob ich allfällige Tasks / Aufgaben entweder komplett streichen kann, oder weniger Zeit für die einplanen könnte. Wir haben uns auch darauf geeinigt, dass ich das Planen für die Dokumentation komplett herausnehme, und dies einfach Teil des täglichen Arbeitens ist

Projekttag 2 (15.03.2023)

	Dario
Geplante Aktivitäten	• Login Page • Registration Page • Board Overview Page • Board Detail Page
Durchgeführte Arbeit	Angefangen habe ich mit dem Login und der Registration. Diese hatte ich schnell erledigt da ich mich schon gut mit angular forms auskenne. Mit dem Backend für das Anmelden und Registrieren habe ich bisschen länger gebraucht. Als dies auch fertig war habe ich der ganze Registration Prozess sowie den Anmelde Prozess getestet und allfällige Kleinigkeiten noch korrigiert die aufgetaucht sind. Danach wurde die Board Overview und Detail Page erstellt. Ich musste mir zuerst ein Konzept überlegen, wie ich diese gerne gestalten will. Ich habe mir ein bisschen Inspiration von Trello geholt und entschieden die Seite auch in Balken / Columns darzustellen.
Erfolge & Misserfolge	Das Frontend wurde etwas schneller fertig da ich schon mehr Erfahrung hatte und wusste, wie ich dies aufbereiten muss. Da ich schon länger kein Backend mehr erstellt habe, musste ich mich zuerst ein bisschen mehr einlesen. Am Ende des Tages wurde ich jedoch mit meinen gewollten Tasks fertig, auch wenn ich den Zeit Aufwand für den Backend Teil ein wenig unterschätzt habe.
Hilfestellung	Am Anfang des Tages haben wir von der Expertin / Modullehrerin noch ein kleiner Input bekommen. Am Tag zuvor mussten wir unsere Arbeitsjournale abgeben, welche sie anschliessend angeschaut hat. Sie hat uns den Tipp gegeben, dass alle ihr Arbeitsjournal besser erstellen / gestallten / ausfüllen sollten. Besonders von der Länge her, wurde gesagt, dass ein Projekttag ungefähr eine ganze Seite lang sein sollte.

Projekttag 3 (17.03.2023)

	Dario
Geplante Aktivitäten	• Board Features implementieren • Ticket Detail Dialog implementieren • Ticket Features implementieren
Durchgeführte Arbeit	Am Morgen habe ich angefangen, meine Aufgaben, die ich noch nicht erledigen konnte, abzuschliessen. Dies beinhaltete das Abschliessen der Board Übersicht. Als alle Aufgaben des vorherigen Projekttages erledigt waren habe ich mich an die Board Features gewandt. Ich habe mir überlegt, wie ich am besten den Invite Code eines Boards anzeigen kann, sowie auch dies nur zugänglich machen kann für Administratoren eines Boards. Die Lösung war ganz einfach. Bei der Board Übersicht wird geschaut, ob der angemeldete Benutzer Admin ist, und wenn dies der Fall ist, wird ein «Settings» Knopf angezeigt welches ein neuer Dialog öffnet in dem der Invite Code angezeigt wird. Später werde ich auch hier in diesem Dialog die Benutzerverwaltung eines Boards implementieren. Danach habe ich mit dem Ticket Detail Dialog angefangen. Ich habe studiert, was alles in einem Ticket definiert werden soll. Dazu gehört der Titel, eine Beschreibung, eine Benutzer Zuweisung, ein Ticket Tag sowie eine Checkliste von Tasks. Die Ticketbeschreibung habe ich mit einem Gratis «Plugin» namens TinyMC erstellt. Dies ist ein Gratis-Plugin, mit welchem man ganz einfach Text definieren und editieren kann. Als der Ticket Detail Dialog fertig war habe ich angefangen die Ticket Features zu implementieren. Teil davon war das Speichern, hinzufügen eines neuen Tickets, Erstellen einer «Task» Liste, das zuweisen von Benutzern sowie das Hinzufügen von Tags. Die Tags wurden mit Angular Material implementiert
Erfolge & Misserfolge	Im Großen und Ganzen war der Tag erfolgreich. Ich konnte vieles Erledigen gemäss Zeitplan, jedoch wurde ich mit den Ticket Features noch nicht ganz fertig. Dies bedeutet das ich am nächsten Projekttag die restlichen Features implementieren muss. Jedoch habe ich dies bereits in meinem Zeitplan mit eingeplant, damit ich noch genügend Zeit habe, diese Aufgabe am nächsten Tag zu implementieren.
Hilfestellung	

Projekttag 4 (21.03.2023)

	Dario
Geplante Aktivitäten	• Ticket Features implementieren • Board Benutzer Verwaltung
Durchgeführte Arbeit	Angefangen habe ich mit den Aufgaben des Vorherigen Tages, sprich mit dem Implementieren der Ticket Features. Teil davon war die Funktionalität der Checkliste. Das Erstellen einer Checkliste funktionierte schon aber das Ankreuzen von Tasks bzw. das markieren eines Tasks als erledigt funktionierte noch nicht. Ich habe dies so implementiert, dass man die Checkliste Task für Task erledigen kann und in der Board Übersicht sieht, wie viele Tasks eines Tickets bereits erledigt wurden. Falls eine Checkliste komplett erledigt war wird in der Board Übersicht ein Grünes Feld auf dem Ticket ersichtlich, damit man weiss, dass diese Person mit dem Ticket bereits fertig ist. Ich habe danach auch noch das Entfernen von bestehenden Tasks erledigt. Als nächstes habe ich begonnen, die Benutzer Verwaltung für ein Board zu erledigen. Die sollte nur für Owners / Admins eines Boards möglich sein. Da ich bereits eine Settings Page für mein Board erstellt habe, auf der man den Invite Code eines Boards sieht, habe ich auf derselben Seite auch die Benutzer Verwaltung eingebaut. Dies war simple da ich dies ähnlich wie die Benutzer Verwaltung eines Tickets implementieren konnte. Das einzige, auf das ich achten musste, ist das ein Admin eines Boards sich selbst nicht von dem Board entfernen kann. Im selben Zug habe ich noch bemerkt, dass man ein Board gar nicht löschen kann. Die habe ich noch schnell mit einem roten Button auf der Settings Page implementiert. Sprich nur ein Admin kann auch ein Board löschen. Im backend musste ich dann noch sicherstellen, dass alle Benutzer, welche Zugriff auf dieses Board hatten, dieses Board nicht mehr in Ihrem Dashboard sehen bzw. kein Zugriff mehr darauf haben.
Erfolge & Misserfolge	Ich wurde mit all meinen geplanten aufgaben fertig und konnte sogar noch ein Feature implementieren, welches mir nicht bewusst war bzw. auf dem Zeitplan fehlte.
Hilfestellung	

Projekttag 5 (22.03.2023)

	Dario
Geplante Aktivitäten	• Bug Fixes die Auftauchen während der Durchführung. • Weiteres Ausfüllen der Dokumentation • Überprüfen ob alle Features eingebaut wurden
Durchgeführte Arbeit	Ich konnte diverse Bug Fixes erledigen. Eines davon war das editieren von einer Checkliste, welches dazu geführt hat, dass sich der «Task Counter» nicht mit neuen werten erweitert hat. Danach habe ich begonnen die Dokumentation weiterhin auszufüllen. Der Planung Teil ist nun so gut wie fertig. Am Nachmittag habe ich mit dem Dokumentieren der Realisierung begonnen. Ich habe zudem noch überprüft ob ich alle geplanten / sinnvollen Features in der Applikation eingebaut habe. Dort habe ich realisiert, dass das Erfassen von Pullrequests und Testrun URLs noch nicht implementiert war. Da dies jedoch keine grosse Aufgabe war konnte ich dies innerhalb 10 Minuten bereits implementieren.
Erfolge & Misserfolge	Da ich bereits am vorherigen Tag mehr implementieren konnte als erwartet, hatte ich heute mehr Zeit, um meine Dokumentation zu erweitern.
Hilfestellung	Zweites Gespräch mit dem Experten. Mit dem Experten zusammen wurde festgestellt, dass das Layout / aussehen des Zeitplans noch Verbesserung Möglichkeiten hat. Es wurde explizit entschieden, dass dies für diese Projektarbeit aber nicht mehr korrigiert werden muss. Es wurde auch erwähnt, dass mein Arbeitsjournal nicht korrekt aufgestellt ist. Es sollte keine Reihe für «Geplante Aktivitäten gemäss Zeitplan» geben, sondern einfach nur eine Reihe: «Geplante Aktivitäten». Dies liegt daran, dass es sehr schnell passieren kann, dass man einige Tage gemäss Zeitplan hinterher ist. Genauso wurde erwähnt, dass meine Reflexionen und Lösungswege nicht gut genug sind bzw. nicht existieren im Arbeitsjournal.

Projekttag 6 (24.03.2023)

	Dario
Geplante Aktivitäten	• Fertigstellen der Dokumentation • Fertigstellen des Zeitplans • Letzten Bug Fixes • Abgabe des Projektes
Durchgeführte Arbeit	Als erstes habe ich die Applikation ein letztes mal ausführlich getestet. Allfällige Fehler wurden noch korrigiert. Dann wurde die Dokumentation fertig geschrieben und der Zeitplan dementsprechend ausgefüllt. Als letztes wurde das Projekt abgegeben.
Erfolge & Misserfolge	Der heutige Tag ist eher etwas stressig da noch einige Punkte in der Dokumentation ausgefüllt werden müssen. Dies habe ich jedoch alles noch innerhalb der angegebener Zeit geschafft. Damit das nächste mal der letzte tag etwas weniger stressig wird werde ich mir vornehmen, genau zu schauen was wirklich alles noch gemacht werden muss da im verlauf des Tages noch einige Sachen aufgetaucht sind, an die ich nicht gedacht habe.
Hilfestellung	

3.1 Versionen der Dokumentation

Datum	Aktivität
V 0.2	Die Grundstruktur für die Dokumentation wurde erstellt. Das Arbeitsjournal für den ersten Tag wurde ausgefüllt
V 0.4	Das Arbeitsjournal für den zweiten Tag wurde ausgefüllt. Das Testkonzept wurde erfasst.
V 0.6	Das Arbeitsjournal für den dritten Tag wurde ausgefüllt.
V 0.8	Das Arbeitsjournal für den vierten Tag wurde ausgefüllt. Das Vorwort sowie die Ausgangslage wurde erfasst und definiert.
V 0.9	Das Arbeitsjournal für den fünften Tag wurde aufgefüllt. Teil der Dokumentation wurde erweitert ausgefüllt.
V 1.0	Finale Version der Dokumentation welche abgegeben wurde.

4 Projekt (Teil 2)

4.1 Kurzzusammenfassung

4.1.1 Ausgangslage

In meinem Betrieb wird mit einem Tool namens Trello gearbeitet. Es hilft beim Planen eines Sprints und man sieht, welche Aufgaben ein Entwickler zu erledigen hat. Man kann Tickets erfassen und diese dementsprechend auch mehreren Entwicklern zuweisen.

Mir ist nach ein paar Jahren aufgefallen, dass dieses Tool gewisse Verbesserungsmöglichkeiten hat. Eines davon ist, dass momentan nicht ersichtlich ist, wenn ein Ticket abgeschlossen wurde, ob bereits ein Pull Request und Testrun für diese Aufgabe existiert. Falls ja, muss man Manuel in der PR-Liste und Testrun Liste suchen gehen, um zu sehen welcher PR zu welchem Ticket gehört.

Es soll nun eine Webapplikation geschaffen werden, in welcher man all diese fehlenden Informationen innerhalb eines Tickets erfassen kann.

Diese Projektarbeit basiert auf keinem vorherigen Projekt und wird deshalb von null auf aufgebaut.

4.1.2 Umsetzung

Das Projekt wurde unter Verwendung der IPERKA-Projektmethodik durchgeführt.

Als Erstes stand die Informationsphase an, in dieser wurde gemäss der Aufgabestellung Anforderungen bzw. individuelle Kriterien definiert. Es wurden auch die generellen Anforderungen an die Applikation bekannt gegeben.

Während der Planungsphase wurde ein Zeitplan definiert und dementsprechend ausgefüllt. Es wurde auch eine Grundstruktur für die Dokumentation erstellt, mit den wichtigsten Überschriften.

In der Realisierungsphase wurde mithilfe des Angular-Webframework basierend auf TypeScript, HTML und CSS das Frontend und mithilfe von .NET Core basierend auf C# das Backend und die API-Schnittstelle entwickelt. Für das Speichern der Daten wurde MongoDB verwendet.

4.1.3 Ergebnis

Das Informatik Projekt, welches durchgeführt wurde, war ein großer Erfolg. Alle geplanten Features, wie das Erfassen von Tickets, das Erstellen von Boards, die Verwaltung von Benutzern, die Definition von Pullrequests und die Definition von Test Runs, wurden erfolgreich implementiert.

Durch die Implementierung von Taskchain wurde das Projektmanagement noch effizienter. Es ist nun einfacher, einen Sprint zu planen und den Fortschritt der einzelnen Tickets und Aufgaben im Auge zu behalten. Durch die Übersichtlichkeit des Boards wird schnell ersichtlich, welche Aufgaben bereits abgeschlossen wurden und welche noch offen sind. Dies erleichterten die Planung und Priorisierung von Aufgaben enorm und trägt zur Steigerung der Produktivität bei.

4.2 Informieren

Das genaue Analysieren der Aufgabenstellung und deren Anforderungen ist das Wichtigste in der ersten Phase der Projektplanungsmethode IPERKA. Das ganze Projekt basiert auf diesem ersten Schritt.

4.2.1 Anforderungen

Hier werden alle funktionalen und nicht funktionalen Anforderungen basierend auf der Aufgabenstellung genau definiert. Basierend auf diesen Anforderungen werden im nächsten Schritt die Implementierungen geplant.

Nicht-Funktionale Anforderungen

- Der Code-Style im Frontend soll formatiert, lesbar und sauber kommentiert sein
- Der Code-Style im Backend soll formatiert, lesbar und sauber kommentiert sein
- Die Passwörter der Benutzer sollten sicher gespeichert werden

Funktionale Anforderungen

- Benutzer können sich mit einem Benutzernamen und Passwort registrieren und einloggen
- Jeder Benutzer kann ein «Board» erstellen, was ihn zu einem Admin dieses Boards macht.
- Administratoren können Benutzer innerhalb eines Boards verwalten
- Tickets können mit Checklisten erfasst werden
- Tickets können einem Benutzer zugewiesen werden
- Tickets, in denen die Checkliste abgeschlossen wurde, werden grün markiert.
- Tickets können in verschiedene Spalten verschoben werden
- Tickets können gelöscht werden
- Der zugewiesene Benutzer kann die Checkliste Bearbeiten
- Der zugewiesene Benutzer kann das Ticket verlassen
- Der zugewiesene Benutzer kann URLs zu aktuellen Test-Runs / Pullrequests hinzufügen.
- Alle Benutzer des Boards können die aktuellen Tickets sehen

4.3 Planen

In dieser Phase wird der Verlauf des Projektes bestimmt und ist somit ein essenzieller Teil in der Projektplanungsmethode IPERKA. In dieser Phase wird der zeitliche Ablauf definiert und in den Zeitplan eingefügt.

Der Zeitplan befindet sich beim Punkt «Geplante Arbeiten». Aus Redundanzgründen wird dieser hier nicht nochmals abgebildet.

Da keine Testabdeckung definiert wurde, werden bei dieser Applikation keine UI / Unit Tests geschrieben. Jedoch wird ein Testkonzept erfasst, welches bestimmte Aspekte / Funktionalitäten der Applikation testet. Diese Tests werden von dem Entwickler in dem Testkonzept definiert und dementsprechend von Hand aus ausgeführt und das Verhalten inklusive Resultat detailliert beschrieben.

4.3.1 Testkonzept

Das folgende Testkonzept soll beschreiben, wie getestet werden muss, sodass die Funktionalität der Applikation korrekt geprüft wird. Die Tests werden auf einem Geschäfts Laptop mit einem Chrome Browser geprüft und detailliert dokumentiert.

Testfall-Nr: 1	Registration eines neuen Benutzers
Beschreibung	Ein neuer Benutzer registriert sich bei der Applikation.

Durchführung	• Der Benutzer öffnet die URL: «<Host>/register»
	• Er füllt das Form aus wie folgt:
	  • Username: Andrin
	  • Password: AndrinTaskchain123!
	  • Confirm Password: AndrinTaskchain123!
	• Der Benutzer klickt auf «Register»

Erwartetes Resultat	Der Benutzer wird auf die «/dashboard» Page weitergeleitet und sieht nur die Optionen «Join Board» und «+ Add Board» sowie in der Navbar sein username und die Optionen «Dashboard» und «Logout»

Testfall-Nr: 2	Registration eines neuen Benutzers mit bereits vorhandenem username
Beschreibung	Ein neuer Benutzer will sich bei der Applikation Registrieren und verwendet ein bereits existierender Benutzername.

Durchführung	• Der Benutzer öffnet die URL: «<Host>/register»
	• Er füllt das Form aus wie folgt:
	  • Username: Philipp
	  • Password: PhilippTaskchain123!
	  • Confirm Password: PhilippTaskchain123!
	• Der Benutzer klickt auf «Register»
	• Der Benutzer klickt auf den «Logout» Knopf in der Navbar
	• Ein neuer Benutzer öffnet die URL: «<Host>/register»
	• Er füllt das Form aus wie folgt:
	  • Username: Philipp
	  • Passwort: Philipp123!
	• Der Benutzer klickt auf «Register»

Erwartetes Resultat	Der erste Benutzer wird auf die «/dashboard» Page weitergeleitet und sieht nur die Optionen «Join Board» und «+ Add Board» sowie in der Navbar sein username und die Optionen «Dashboard» und «Logout» Der zweite Benutzer sieht eine rote Meldung «Username is taken» beim Username Input Feld. Bei dem Klick auf «Register» geschieht nichts.

Testfall-Nr: 3	Registration eines neuen Benutzers mit keinem Benutzernamen
Beschreibung	Ein neuer Benutzer will sich bei der Applikation Registrieren. Jedoch klickt der Benutzer aus Versehen auf «Registrieren» Ohne das ein User Namen anzugeben

Durchführung	• Der Benutzer öffnet die URL: «<Host>/register»
	• Er füllt die Form aus wie folgt:
	  • Username: «leer»
	  • Passwort: KuSdi123!
	• Der Benutzer klickt auf Registrieren

Erwartetes Resultat	Eine Meldung taucht auf in welcher steht, dass der Benutzername angegeben werden muss.

Testfall-Nr: 4	Login eines Benutzers
Beschreibung	Ein bestehender Benutzer meldet sich bei der Applikation an.

Durchführung	• Der Benutzer öffnet die URL: «<Host>/login»
	• Er füllt das Form aus wie folgt:
	  • Username: Andrin
	  • Passwort: AndrinTaskchain123!
	• Der Benutzer klickt auf «Log In»

Erwartetes Resultat	Der Benutzer wird auf die «/dashboard» Page weitergeleitet und sieht die Optionen «Join Board» und «+ Add Board» sowie in der Navbar sein username und die Optionen «Dashboard» und «Logout». Der Benutzer hat noch keine Boards und sieht diese deshalb auch noch nicht.

Testfall-Nr: 5	Login eines Benutzers mit falschen Anmeldedaten
Beschreibung	Ein bestehender Benutzer meldet sich bei der Applikation mit falschen Anmeldedaten an.

Durchführung	• Der Benutzer öffnet die URL: «<Host>/login»
	• Er füllt das Form aus wie folgt:
	  • Username: Andrinn
	  • Passwort: AndrinTaskchain123!
	• Der Benutzer klickt auf «Log In»

Erwartetes Resultat	Eine Meldung erscheint, welche der Benutzer darauf hinweist, dass die angegebenen Daten kontrolliert werden sollen.

Testfall-Nr: 6	Login eines Benutzers mit fehlendem Username
Beschreibung	Ein bestehender Benutzer meldet sich bei der Applikation an, ohne ein Username anzugeben.

Durchführung	• Der Benutzer öffnet die URL: «<Host>/login»
	• Er füllt das Form aus wie folgt:
	  • Username:
	  • Passwort: AndrinTaskchain123!
	• Der Benutzer klickt auf «Log In»

Erwartetes Resultat	Eine Meldung erscheint, welche der Benutzer darauf hinweist, dass der Username angegeben werden soll.

Testfall-Nr: 7	Ein Benutzer meldet sich an und navigiert auf eine Board Übersicht
Beschreibung	Ein bestehender Benutzer meldet sich bei der Applikation an und navigiert zur Übersicht eines Boards.

Durchführung	• Der Benutzer öffnet die URL: «<Host>/login»
	• Er füllt das Form aus wie folgt:
	  • Username: Andrin
	  • Passwort: AndrinTaskchain123!
	• Der Benutzer klickt auf «Log In»
	• Auf der Dashboard Seite klickt der Benutzer auf das erste aufgelistete Board.

Erwartetes Resultat	Der Benutzer wird auf die Board Übersicht Seite weitergeleitet und sieht somit (falls vorhanden) Seine Spalten und Tickets

Testfall-Nr: 8	Ein angemeldeter Benutzer erstellt ein neues Board
Beschreibung	Ein angemeldeter Benutzer erstellt ein neues Board.
Durchführung	• Der Benutzer öffnet die URL: «<Host>/dashboard» • Auf der Dashboard Seite klickt der Benutzer auf «+ Add Board». • Auf dem aufgetauchtem Dialog gibt der Benutzer ein Board Titel: «Entwicklung» ein. • Der Benutzer klickt auf «Save»
Erwartetes Resultat	Der Benutzer wird auf die Board Übersicht Seite weitergeleitet, auf der er den eingegebenen Titel und daneben einen Button «Settings» sieht.

Testfall-Nr: 9	Ein angemeldeter Benutzer erstellt ein neues Board, ohne ein Titel anzugeben
Beschreibung	Ein bestehender Benutzer meldet sich bei der Applikation an und erstellt ein neues Board.
Durchführung	• Der Benutzer öffnet die URL: «<Host>/dashboard» • Auf der Dashboard Seite klickt der Benutzer auf «+ Add Board». • Auf dem aufgetauchtem Dialog gibt der Benutzer keinen Board Titel ein. • Der Benutzer klickt auf «Save»
Erwartetes Resultat	Es taucht ein Dialog auf, welcher der Benutzer darauf hinweist, dass ein Titel angegeben werden muss.

Testfall-Nr: 10	Ein angemeldeter Benutzer erstellt ein neues Ticket
Beschreibung	Ein bestehender Benutzer meldet sich bei der Applikation an und erstellt ein neues Ticket.

Durchführung	• Der Benutzer öffnet die Detail Ansicht eines bestehendes Board
	• In der ersten Spalte klickt der Benutzer unten auf +Add Card
	• Der Benutzer füllt das Ticket wie folgt aus:
	  • Titel: Benutzer Registration
	  • Im Editor: TODOS werden noch definiert.
	• Der Benutzer klickt auf «Save»

Erwartetes Resultat	Der Dialog schliesst sich und das Ticket wird in der entsprechender Spalte angezeigt. Der angegebene Titel ist ersichtlich

Testfall-Nr: 11	Ein angemeldeter Benutzer editiert ein Ticket
Beschreibung	Ein bestehender Benutzer editiert ein Ticket.
Durchführung	• Der Benutzer öffnet die Detail Ansicht eines bestehendes Board • In der ersten Spalte klickt der Benutzer auf ein bestehendes Ticket • Der Benutzer editiert das Ticket wie folgt: • Titel: Benutzer Login • Der Benutzer klickt auf «Save»
Erwartetes Resultat	Der Dialog schliesst sich und der neue Titel ist auf dem Ticket ersichtlich.

Testfall-Nr: 12	Ein angemeldeter Benutzer fügt sich selbst einem Ticket hinzu
Beschreibung	Ein angemeldeter Benutzer fügt sich selbst einem Ticket hinzu
Durchführung	• Der Benutzer öffnet die Detail Ansicht eines bestehendes Board • In der ersten Spalte klickt der Benutzer auf ein bestehendes Ticket • Der Benutzer klickt auf Add • Der Benutzer klickt auf dem aufgetauchtem Dialog auf den hinzufügen Button neben seinem Benutzernamen
Erwartetes Resultat	Der Dialog schliesst sich und der Benutzer ist in der Benutzer Liste auf dem Ticket ersichtlich.

Testfall-Nr: 13	Ein angemeldeter Benutzer joined einem bestehendem Board mit einem Invite Code
Beschreibung	Ein angemeldeter Benutzer joined einem bestehendem Board mit einem Invite Code
Durchführung	• Der Benutzer klickt auf «Join Board» in seinem Dashboard. • Der Benutzer gibt einen Invite Code, an den er von einer Person die Owner eines boards ist, erhalten hat. • Der Benutzer klickt auf «Join»
Erwartetes Resultat	Der Benutzer wird auf die Board Übersicht weitergeleitet.

Testfall-Nr: 14	Ein angemeldeter Benutzer gibt einen Invaliden Invite Code an
Beschreibung	Ein angemeldeter Benutzer gibt einen Invaliden Invite Code an welcher zu lange ist
Durchführung	• Der Benutzer klickt auf «Join Board» in seinem Dashboard. • Der Benutzer gibt den Invite Code: OKIHVB7TI ein
Erwartetes Resultat	Auf dem Formular wird bei dem Invite Code Feld eine rote Meldung angezeigt die beschreibt, dass der Invite Code Invalid ist.

4.3.2 Sitemap

Diese Sitemap soll beim Realisieren einen guten Überblick über die verschiedenen Seiten und deren Relationen verschaffen. Sie zeigt somit die logische Struktur der Webapplikation auf.

- **Login**
  - Der Benutzer kann sich hier in seinen Account einloggen
- **Register**
  - Ein neuer Benutzer kann sich hier ein Account erstellen
- **Dashboard**
  - Ein Benutzer sieht hier all seine Boards und kann auch ein neues Erstellen oder auf die Seite Join Board gehen
- **Join Board**
  - Ein Benutzer kann ein bestehendes Board mithilfe eines «Invite Code's» beitreten
- **Add Board**
  - Ein Benutzer kann hier ein neues Board erstellen
- **Board Übersicht**
  - Ein Benutzer sieht hier die Übersicht des ausgewählten Boards inklusive alle Tickets
- **Board Settings** (Nur für User mit «Owner» Rechten zugänglich)
  - Ein Besitzer eines Boards kann hier die Benutzer verwalten, sowie auf den Invite Code zugreifen, welcher mit anderen Benutzern Teilen kann, damit diese dem aktuellen Board beitreten können
- **Ticket Detail**
  - Hier kann ein Benutzer ein Ticket Bearbeiten / ausfüllen. Es kann ein Titel, eine Beschreibung, eine Checkliste und Tags erfasst werden. Es ist auch ersichtlich, welche Benutzer diesem Ticket zugewiesen sind.
- **Ticket Detail Benutzer Verwaltung**
  - Ein Benutzer sieht hier eine Liste aller Benutzer des Boards und kann aussuchen welche Benutzer diesem Ticket zugeteilt werden sollen.
- **Ticket Detail Tag Farben**
  - Ein Benutzer kann hier aussuchen, welche Farbe ein Tag hat.

4.4 Entscheiden

In der dritten Phase der IPERKA Projektplanungsmethode wird entschieden, welcher Lösungsweg verwendet wird. Dabei werden Lösungsvarianten verglichen und allfällige Risiken evaluiert.

4.4.1 Varianten

Da wir laut Anforderungen selbst entscheiden konnten welche Technologien verwendet werden, musste ich zuerst herausfinden, welche Frameworks und welche Datenbanken in Frage kommen würden. Entweder stelle ich mir der Herausforderung, ein komplett neues Framework und eine neue Datenbank zu verwenden, mit der ich noch nie wirklich etwas zu tun hatte und deshalb auch keine Erfahrung damit habe, oder nicht.

Für das Frontend hatte ich die Optionen Vue, React oder Angular zu verwenden. Für die Datenbank standen SQL und MongoDB infrage.

4.4.2 Kriterien

Das Risiko mit einer SQL-Datenbank ist, dass ich erstens nicht viel Ahnung davon haben und ich nicht weiss, wie ich es in mein Projekt einbinden könnte und zweitens wusste ich, dass ich mit einer SQL-DB mit dem Speichern von bestimmten Daten, später Probleme haben werde. Es war klar das SQL jedoch eine bessere Variante für das Speichern von persönlichen Daten wäre.

Mit der MongoDB hatte ich schon ein bisschen Erfahrung und wusste, dass persönliche Daten genauso gut in einer NoSQL-DB gespeichert werden können. Ich wusste auch, dass mir das Speichern von Objekten in einer MongoDB einfacher fallen wird.

Das Risiko bei React und Vue waren, dass dies viel mehr Aufwand für mich sein wird, da noch ich noch nie damit gearbeitet habe. Das bedeutet, dass ich zuerst einmal diese Frameworks verstehen und erlernen muss und dann eine komplette funktionale Applikation damit gestalten sollte. Dies kann dazu führen, dass ich meine Funktionale und nicht Funktionale Anforderungen nur zu einem gewissen Mass erfüllen könnte, da ich zuerst viel Zeit in das Erlernen des Frameworks einsetzen müsste.

4.4.3 Entscheid

Aufgrund der Vorkenntnisse und der möglichen Kriterien habe ich mich schlussendlich für die Kombination aus Angular, .NET Core (C#) und der MongoDB entschieden. Wie bereits bei den Varianten erwähnt wurde, habe ich MongoDB und .NET Core (C#) am meisten bei Schulprojekten verwendet und bin deshalb auch am fittesten damit. Ebenfalls liegt für mich bei diesem Projekt der Fokus auf Qualität und Funktionalität der Applikation, sowie dem Erlernen wie man korrekt eine Dokumentation führt.

4.5 Realisieren

In der vierten Phase der Projektmethode IPERKA geht es um die Umsetzung und Realisierung der Applikation. Hier wird ein Teil der komplexeren Elemente der Realisierung erklärt und dargestellt. Der Code wird mit sinnvollen Kommentaren erweitert welches der Verständlichkeit sehr helfen kann.

4.5.1 Frontend

Wie bereits erwähnt wurde das Frontend der Applikation mit Angular erstellt. Angular ist ein Front-End-TypeScript-Framework, das für die Erstellung von Webanwendungen verwendet wird. Es wurde von Google entwickelt und wird derzeit von einer Gemeinschaft von Entwicklern gepflegt. Das Projekt ist Open Source und der Quellcode ist auf GitHub zu finden.

Angular verwendet eine komponentenbasierte Architektur, das heisst es ermöglicht Entwicklern, Anwendungen zu erstellen, indem sie sie in kleinere, wiederverwendbare Komponenten aufteilen. Dadurch lassen sich komplexe Anwendungen leichter entwickeln und pflegen. Die ganze Struktur der Applikation ist so viel organisierter und viel übersichtlicher.

Angular wird mit TypeScript erstellt, einer Obermenge von JavaScript, die optionale statische Typisierung und andere Funktionen bietet. Dadurch sind Angular-Anwendungen einfacher zu schreiben und zu warten, insbesondere bei umfangreichen Projekten.

Angular-Anwendungen werden in der Regel mit einer Kombination aus HTML, CSS und TypeScript erstellt. Es umfasst eine Reihe von Tools und Bibliotheken, die das Erstellen und Testen von Webanwendungen erleichtern, sowie eine Vielzahl von Funktionen wie Routing, Formularvalidierung und Dependency Injection zur Verfügung stellt.

Angular bieten auch eine umfangreiche Dokumentation und eine Fülle von Tutorials und Beispielen, die es Entwicklern erleichtern, mit dem Framework zu beginnen und ihre Fähigkeiten zu verbessern. Darüber hinaus wird Angular kontinuierlich weiterentwickelt und aktualisiert, um mit den neuesten Entwicklungen im Web-Technologiebereich Schritt zu halten. Die Community ist sehr aktiv und es gibt viele Foren und Community-basierte Support-Gruppen, in denen Entwickler Fragen stellen und Antworten finden können.

Zusammenfassend lässt sich sagen, dass Angular eine leistungsstarke, flexible und leicht erlernbare Technologie ist, die für die Erstellung von Webanwendungen geeignet ist. Durch die Verwendung von Angular können Entwickler Anwendungen erstellen, die skalierbar, wartbar und robust sind.

Struktur:

Da eine übersichtliche Ordner Struktur angestrebt wurde habe ich verschiedene Stufen dafür erstellt. Die Services und Models befinden sich auf der äussersten Ebene innerhalb des shared Ordners. Zusätzlich habe ich eine extensions Klasse erstellt welche mir globale Funktionen zur verfügung stellt.

Innerhalb des Components Ordners befinden sich alle Seiten wie Dashboard, Login, Registration und die Page-Not-Found Seite. Innerhalb des board Ordners sind alle Sub-Pages, die Teil des Boards und Tickets Verwaltung sind.

Dank solch einer Struktur wurde eine bessere Übersicht erschaffen und man weiss immer, wo sich was befindet.

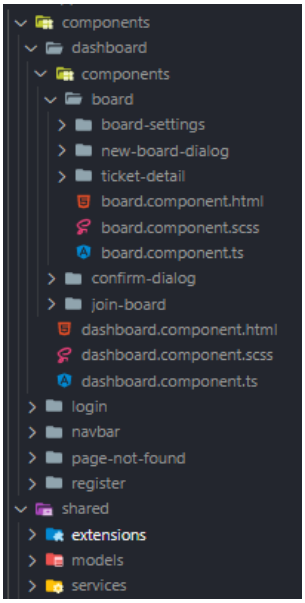

Abbildung 2 - Frontend Ordner Struktur

Design

Für das Design hatte ich nicht eine Idee. Ich habe keine Mockups erstellt, sondern einfach losprogrammiert. Da mein Projekt ein bisschen oder ziemlich ähnlich ist wie Trello, hatte ich einen sehr guten Anhaltspunkt .

Für die Farben entscheidung habe ich die website: https://coolors verwendet. Dies ist ein Online Tool, mit welchem man verschiedenste Farbpaletten zusammenstellen kann.

Hier sehen sie die Farben für die bei Taskchain verwendet wurden.

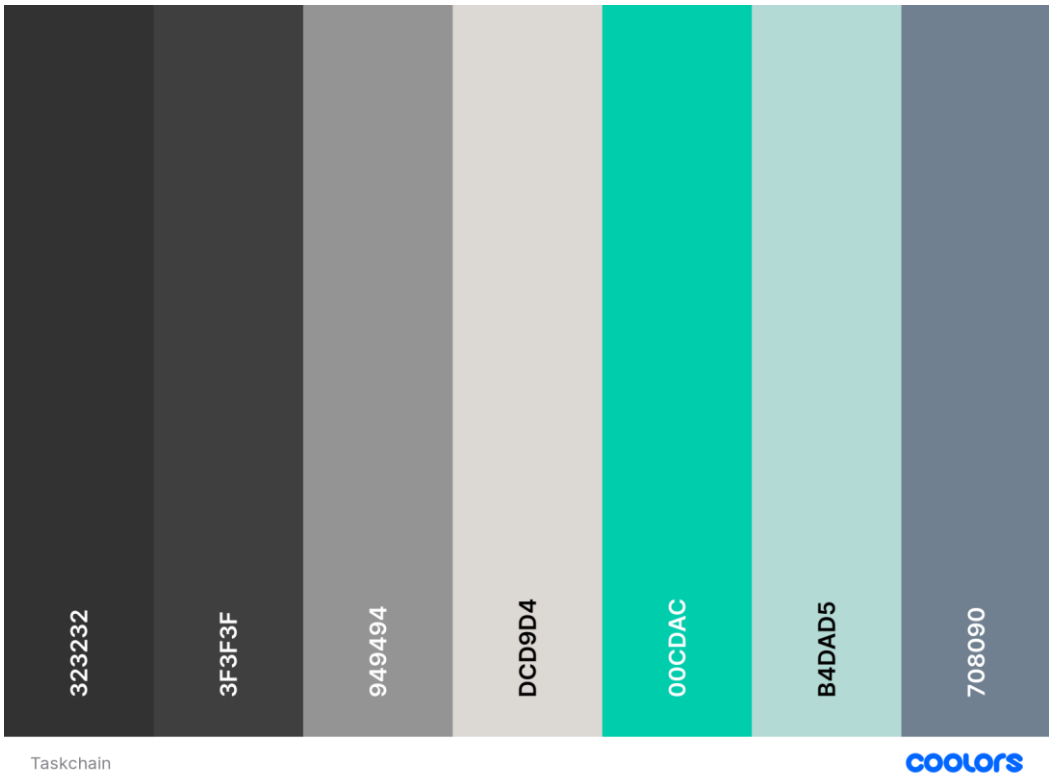

Abbildung 3 - Farbpalette Taskchain

Load Board

Diese Methode ist Teil der Komponente namens "BoardComponent". Das Ziel der Methode "loadBoard()" ist es, das aktuelle Board zu laden, das in der URL-Query-Parameter "id" spezifiziert ist.

Der erste Schritt der Methode ist das Abrufen des Query-Parameter "id" aus der URL mittels "URLSearchParams" und Speicherung des Ergebnisses in der Variablen "boardId". Anschließend wird überprüft, ob der Benutzer Berechtigungen hat, auf das angeforderte Board zuzugreifen. Wenn der Benutzer keine Berechtigungen hat und das angeforderte Board nicht das Board mit der ID "0" ist, wird eine Snackbar geöffnet, die den Benutzer darüber informiert, dass er keine Berechtigung hat, auf das Board zuzugreifen, und ihm die Möglichkeit gibt, auf das Dashboard zurückzukehren.

Wenn der Benutzer jedoch Berechtigungen hat, wird eine Anforderung an den Server gesendet, um die Board-Daten abzurufen. Dies erfolgt mithilfe des "boardService", der in dieser Komponente verfügbar sein muss, um das Board abzurufen.

Wenn das Laden des Boards erfolgreich ist, wird das Ergebnis in der Variablen "board" gespeichert und das UI wird dementsprechend angepasst.

Andernfalls wird eine Snackbar geöffnet, die den Benutzer darüber informiert, dass das Laden des Boards fehlgeschlagen ist, und ihm die Möglichkeit gibt, es erneut zu versuchen.

```ts
/**
   * Load the current board
   * @memberof BoardComponent
   */
  public loadBoard(): void {
    const urlParams = new URLSearchParams(window.location.search);
    const boardId = urlParams.get('id')?.toString();
    if (
      !this.extensions.getUser().boards.find((board) => board.id === boardId) &&
      boardId !== '0'
    ) {
      const ref = this.snackBar.open(
        'You do not have permission to access this board',
        'close',
        {
          horizontalPosition: 'right',
          verticalPosition: 'bottom',
        }
      );

      ref.onAction().subscribe(() => {
        window.location.href = '/dashboard';
      });
    } else if (boardId && boardId !== '' && boardId !== '0') {
      const request: IGetBoardRequest = {
        BoardId: boardId,
      };

      this.boardService
        .getBoard(request)
        .pipe(
          tap((res) => {
            this.board = res;
          }),
          catchError((error) => {
            const ref = this.snackBar.open('Loading Board', 'retry', {
              horizontalPosition: 'right',
              verticalPosition: 'bottom',
            });

            ref.onAction().subscribe((res) => {
              this.loadBoard();
            });

            return error;
          })
        )
        .subscribe();
    } else if (boardId === '0') {
      const newBoardRef = this.dialog.open(NewBoardDialogComponent, {
        data: new BoardModel(),
        disableClose: true,
        maxHeight: '90vh',
        autoFocus: '__non_existing_element__',
      });

      newBoardRef.afterClosed().subscribe((newBoard: BoardModel) => {
        if (newBoard) {
          this.board = newBoard;
          this.createBoard();
        }
      });
    }
  }
```

Abbildung 4 - Load Board Methode

Wenn der Wert von "boardId" gleich "0" ist, wird ein Dialog geöffnet, um ein neues Board zu erstellen. Wenn der Benutzer ein neues Board erstellt, wird das Ergebnis in der Variablen "newBoard" gespeichert und die Methode "createBoard()" aufgerufen, um das Board zu erstellen.

Insgesamt ist diese Methode dafür verantwortlich, das aktuelle Board zu laden, entweder indem es abgerufen wird oder indem es erstellt wird, falls es sich um ein neues Board handelt. Es prüft auch, ob der Benutzer Berechtigungen hat, auf das Board zuzugreifen, und informiert den Benutzer darüber, falls dies nicht der Fall ist.

Validate password

Die Methode "validatePassword()" gehört zum RegisterComponent und validiert das eingegebene Passwort.

In der Methode wird zuerst das Passwort aus dem "userRegister" Objekt extrahiert und in der Variablen "userPassword" gespeichert. Danach folgt eine Reihe von Bedingungen, die überprüfen, ob das Passwort bestimmten Kriterien entspricht oder nicht. Wenn das Passwort nicht den Anforderungen entspricht, wird die entsprechende Fehlermeldung in der Variablen "validationMessage" gespeichert welche dann im UI angezeigt wird. Die Bedingungen lauten wie folgt:

```
/**
  * Validates the entered password with the confirmation password
  * @memberof RegisterComponent
  */
public validatePassword(): void {
  var userPassword = this.userRegister.Password;
  if (userPassword === '') {
    this.validationMessage = 'required';
  } else if (userPassword.length < 6) {
    this.validationMessage = 'password is too short';
  } else if (userPassword.length > 50) {
    this.validationMessage = 'password is too long';
  } else if (userPassword.search(/\d/) == -1) {
    this.validationMessage = 'password must include a number';
  } else if (userPassword.search(/[a-zA-Z]/) == -1) {
    this.validationMessage = 'password has no letters';
  } else if (
    userPassword.search(/[^a-zA-Z0-9\!\@\#\$\%\^\&\*\(\)\_\+\.\,\;\:]/) != -1
  ) {
    this.validationMessage = 'incorrect characters';
  } else {
    this.validationMessage = 'good to go!';
  }

  if (this.userRegister.Password === this.userRegister.ConfirmPassword) {
    this.passwordsMatch = true;
  } else {
    this.passwordsMatch = false;
  }
}
```

Abbildung 5 - Validate Password Methode

- Wenn das Passwort leer ist, wird "required" in der Variablen "validationMessage" gespeichert.
- Wenn das Passwort weniger als 6 Zeichen lang ist, wird "password is too short" in der Variablen "validationMessage" gespeichert.
- Wenn das Passwort mehr als 50 Zeichen lang ist, wird "password is too long" in der Variablen "validationMessage" gespeichert.
- Wenn das Passwort keine Ziffer enthält, wird "password must include a number" in der Variablen "validationMessage" gespeichert.
- Wenn das Passwort keine Buchstaben enthält, wird "password has no letters" in der Variablen "validationMessage" gespeichert.
- Wenn das Passwort unerlaubte Zeichen enthält, wird "incorrect characters" in der Variablen "validationMessage" gespeichert.
- Wenn das Passwort allen Anforderungen entspricht, wird "good to go!" in der Variablen "validationMessage" gespeichert.

Anschließend wird überprüft, ob das Passwort und das Bestätigungspasswort übereinstimmen. Wenn ja, wird die Variable "passwordsMatch" auf "true" gesetzt, andernfalls auf "false".

Insgesamt stellt die Methode eine grundlegende Validierung für ein Passwortfeld dar, um sicherzustellen, dass das eingegebene Passwort bestimmten Kriterien entspricht und mit dem Bestätigungspasswort übereinstimmt.

4.5.2 Backend

Das Backend wurde mit C# und dem Framework ASP .NET CORE 6.0 erstellt. .NET Core ist eine plattformübergreifende, Open-Source- und modulare Version des .NET-Frameworks. Es wurde von Microsoft entwickelt, um Entwicklern die Erstellung von Anwendungen zu ermöglichen, die auf mehreren Plattformen ausgeführt werden können, darunter Windows, MacOS und Linux.

Eines der wichtigsten Merkmale von .NET Core ist, dass es modular ist, das heisst man kann wählen, welche Komponenten und Bibliotheken man in der Anwendung einbinden möchten. So kann man leichtgewichtige und effiziente Anwendungen erstellen, die nur die Funktionen enthalten, die man benötigt. .NET Core enthält auch eine Reihe von Leistungsverbesserungen und neuen Funktionen im Vergleich zum vollständigen .NET-Framework. Es hat eine schnellere Startzeit und unterstützt Funktionen wie die Side-by-Side-Installation, die es Ihnen ermöglicht, mehrere Versionen von .NET Core auf demselben Rechner ohne Konflikte zu installieren.

Struktur

Wie auch beim Frontend strebe ich im Backend eine gute Ordnerstruktur an, für die leichte Erweiterbarkeit und bessere Organisation und Übersicht.

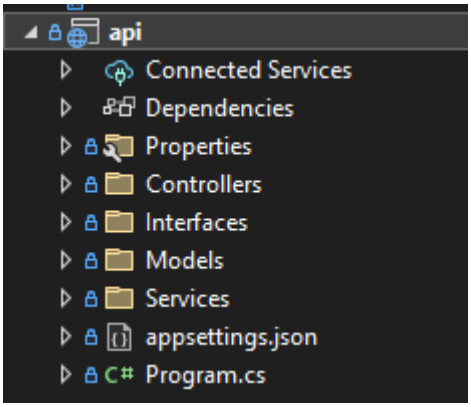

Abbildung 6 - Ordner Struktur Backend

Die Hauptfunktionalitäten bestehen aus Controllers, Models, Services und Interfaces. In den Controllern werden die verschiedenen Routes definiert. Dafür greift er auf die Funktionalitäten der Services zu.

Innerhalb des Models-Ordners werden alle notwendigen Models definiert, die im Backend verwendet werden. Der Ordner enthält auch zwei weitere Unterverzeichnisse, in denen Request- und Response-Models definiert werden. Diese Models werden verwendet, um die Daten zu repräsentieren, die vom Frontend in einer Anfrage gesendet werden und um festzulegen, welche Daten in einer Antwort zurückgesendet werden.

Controller

Der Controller ist für die Routes und Antworten der Requests verantwortlich.

Ich habe mir dafür als Beispiel die Funktion für das löschen eines Boards genommen.

```
[HttpPost]
[Route("deleteBoard")]
0 references
public IActionResult DeleteBoard(DeleteBoardRequest request)
{
    BoardService.DeleteBoard(new Guid(request.BoardId));

    foreach (User user in request.Users)
    {
        UserService.RemoveUserFromBoard(new Guid(user.Id), new Guid(request.BoardId));
    }

    return Ok();
}
```

Abbildung 7 - Delete Board Methode

Diese Methode ist eine ASP.NET Core-Controller-Action-Methode, die über eine HTTP-POST-Anforderung aufgerufen wird und über das Attribut «Route» "deleteBoard" erreichbar ist.

Die Methode nimmt ein Objekt vom Typ "DeleteBoardRequest" als Parameter entgegen und gibt ein "IActionResult"-Objekt zurück. "IActionResult" ist eine Schnittstelle, die eine Aktion in ASP.NET Core darstellt, die eine Antwort an den Client sendet.

In der Methode wird zuerst die "DeleteBoard"-Methode des "BoardService" aufgerufen und mit dem "BoardId" als Parameter aufgerufen, um das Board mit der angegebenen Id aus der Datenbank zu löschen. Die Id wird aus dem Request Objekt von dem Typ «DeleteBoardRequest» herausgelesen. Diese Daten werden von dem Frontend mit einem POST request innerhalb des Body's mitgeschickt.

Dann wird eine foreach-Schleife verwendet, um jeden User, der Teil dieses Boards ist, zu durchlaufen, und die "RemoveUserFromBoard"-Methode des "UserService" aufgerufen, um den User von dem Board zu entfernen. Die "RemoveUserFromBoard"-Methode nimmt die "Id" des Users und die "BoardId" als Parameter und entfernt den User von dem Board.

Schließlich wird die "Ok"-Methode aufgerufen, um eine HTTP 200 OK-Antwort an den Client zurückzugeben, um anzuzeigen, dass die Operation erfolgreich abgeschlossen wurde.

Falls ein fehler während dem ausführen dieser Methode auftritt, wird an den client ein 500-Serverfehler zurückgeschickt. Dass kann der Client abfangen und dementsprechende Fehlermeldungen anzeigen.

Service

Die Services sind die Klassen, die hinter dem Controller die Daten aus der Datenbank holen, hinzufügen oder auch editieren, sie aufbereiten und dann an den Controller schicken. Sie sind Teil der Business Logik und sind die Hauptfunktionalität eines backends.

Hierzu habe ich die Funktion, welche ein neues Board erstellt, ausgewählt und ausführlich beschrieben.

```
// Method for creating a new Board with given title and user as owner
2 references
public Board? CreateBoard(string boardTitle, User user)
{
    try
    {
        MongoDbBoard mongoDbBoard = new MongoDbBoard(new Board() { Title = boardTitle });
        mongoDbBoard.Owner = user.Id; // Set board owner as the given user

        bool inviteCodeExists = true;
        do
        {
            // Generate a 6-character random string to use as the board's invite code
            mongoDbBoard.InviteCode = RandomString(6);

            // Check if a board with the generated invite code already exists, if not, set inviteCodeExists to false to exit the loop
            if (GetBoardByInviteCode(mongoDbBoard.InviteCode) == null)
            {
                inviteCodeExists = false;
            }
        } while (inviteCodeExists); // Loop until a unique invite code is generated

        MongoCRUD.InsertRecord(collection, mongoDbBoard);

        return new Board(mongoDbBoard);
    }
    catch
    {
        // If an exception is thrown during the creation process, return null
        return null;
    }
}
```

Abbildung 8 - Create Board Methode

Diese Methode wird von einem Controller aufgerufen. Sie erstellt ein neues Board in der Datenbank mit einem gegebenen Titel und einem gegebenen Benutzer als Eigentümer. Der Rückgabewert ist eine Board-Instanz oder null, wenn während des Erstellungsprozesses eine Ausnahme auftritt.

Zunächst wird eine neue Instanz der Klasse MongoDbBoard erstellt, indem ein neues Board-Objekt mit dem gegebenen Titel erstellt wird. Der Board-Ersteller, der den Board-Owner darstellt, wird als der gegebene Benutzer festgelegt. Dann wird eine zufällige 6-stellige Zeichenfolge als Einladungscode für das Board generiert. Wenn ein Board mit dem generierten Einladungscode bereits existiert, wird ein neuer Einladungscode generiert, bis ein eindeutiger Einladungscode gefunden wird.

Sobald ein eindeutiger Einladungscode gefunden wurde, wird das neue Board in der Datenbank erstellt, indem die Methode "InsertRecord" der Klasse MongoCRUD aufgerufen wird. Diese Methode gibt dem Board eine einzigartige «ID».

Wenn ein Fehler während des Erstellungsprozesses auftritt, wird eine Ausnahme ausgelöst und null zurückgegeben. Der Controller, der diese Methode verwendet, muss dementsprechend mit dem null wert eine Fehlermeldung an das Frontend liefern, damit eine Fehlermeldung für den Benutzer angezeigt werden kann.

Random Invite Code

Die Methode «RandomInviteCode(int length)» generiert einen zufälligen String mit der Länge, die als Parameter übergeben wird. Sie wird verwendet, um einem Board einen Zufälligen einladungs code zu generieren, welcher Benutzer verwenden können, um dem Board beizutreten.

Der zufällige String besteht aus den Zeichen, die in der Zeichenfolge "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789" definiert sind.

Zunächst wird eine Instanz der Random-Klasse erstellt. Diese Klasse bietet Methoden zum Generieren von Zufallszahlen. Dann wird eine Konstante Zeichenfolge definiert, die alle möglichen Zeichen enthält, die im zufälligen String verwendet werden können.

Die Methode verwendet die Enumerable Repeat-Methode, um die Zeichenfolge chars x-mal zu wiederholen und in einem Array zu speichern. Mit der Select-Methode wird dann für jedes Element des Arrays ein zufälliger Index generiert, um das zufällige Zeichen auszuwählen. Das Ergebnis wird dann als Zeichenfolge zurückgegeben.

```
// Generates a random invite code of a given length.
1 reference
private static string RandomInviteCode(int length)
{
    // Initialize a new instance of the Random class to generate a random number.
    Random random = new();

    // Define a string containing all possible characters.
    const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

    // Generate a random string of the given length by selecting random characters from the chars string.
    return new string(Enumerable.Repeat(chars, length)
        .Select(s => s[random.Next(s.Length)]).ToArray());
}
```

Abbildung 9 - Random Invite Code Methode

4.5.3 Klassendiagramm

Ein Klassendiagramm ist eine grafische Darstellung der Klassen, Interfaces und ihrer Beziehungen in einem Softwaresystem. Es dient dazu, die Struktur und die statischen Beziehungen zwischen den Klassen und deren Methoden und Attributen zu visualisieren.

In einem Klassendiagramm werden die Klassen als Rechtecke dargestellt, die die Klassenname enthalten. Die Methoden und Attribute der Klassen werden als Zeilen in den Rechtecken dargestellt. Die Beziehungen zwischen den Klassen können durch Linien dargestellt werden, die die Abhängigkeiten, Assoziationen, Vererbungen und Aggregationen anzeigen.

Klassendiagramme werden in der Softwareentwicklung verwendet, um die Struktur eines Systems zu visualisieren und zu analysieren. Sie helfen Entwicklern, einen Überblick über die Klassen und ihre Beziehungen zu gewinnen, bevor sie mit der Implementierung beginnen. Dadurch können mögliche Probleme und Designfehler frühzeitig erkannt und behoben werden. Klassendiagramme dienen auch als Dokumentation für das Softwaresystem und können in der Zusammenarbeit zwischen Entwicklern verwendet werden.

Bei Taskchain kann ein Benutzer beliebig viele Boards erstellen. Ein Benutzer kann auch einem Board Beitreten. Ein Board kann beliebig viele Mitglieder haben. Es kann auch beliebig viele Spalten haben. Eine Spalte kann genauso beliebig viele Tickets beinhalten. Ein Ticket kann keine bis zu beliebig viele Mitglieder haben. Es kann auch keine bis zu beliebig viele «Tasks» welche in der Checkliste aufgelistet werden, haben. Ein Ticket kann keine bis zu 3 Tags haben.

Die Klassendiagramme bei Taskchain sehen wie folgt aus:

MongoDb User:

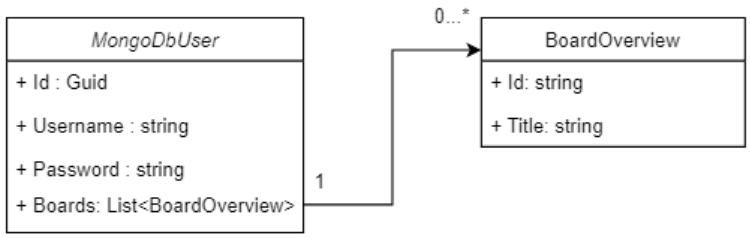

Abbildung 10 - MongoDb User Klassendiagramm

MongoDB Board:

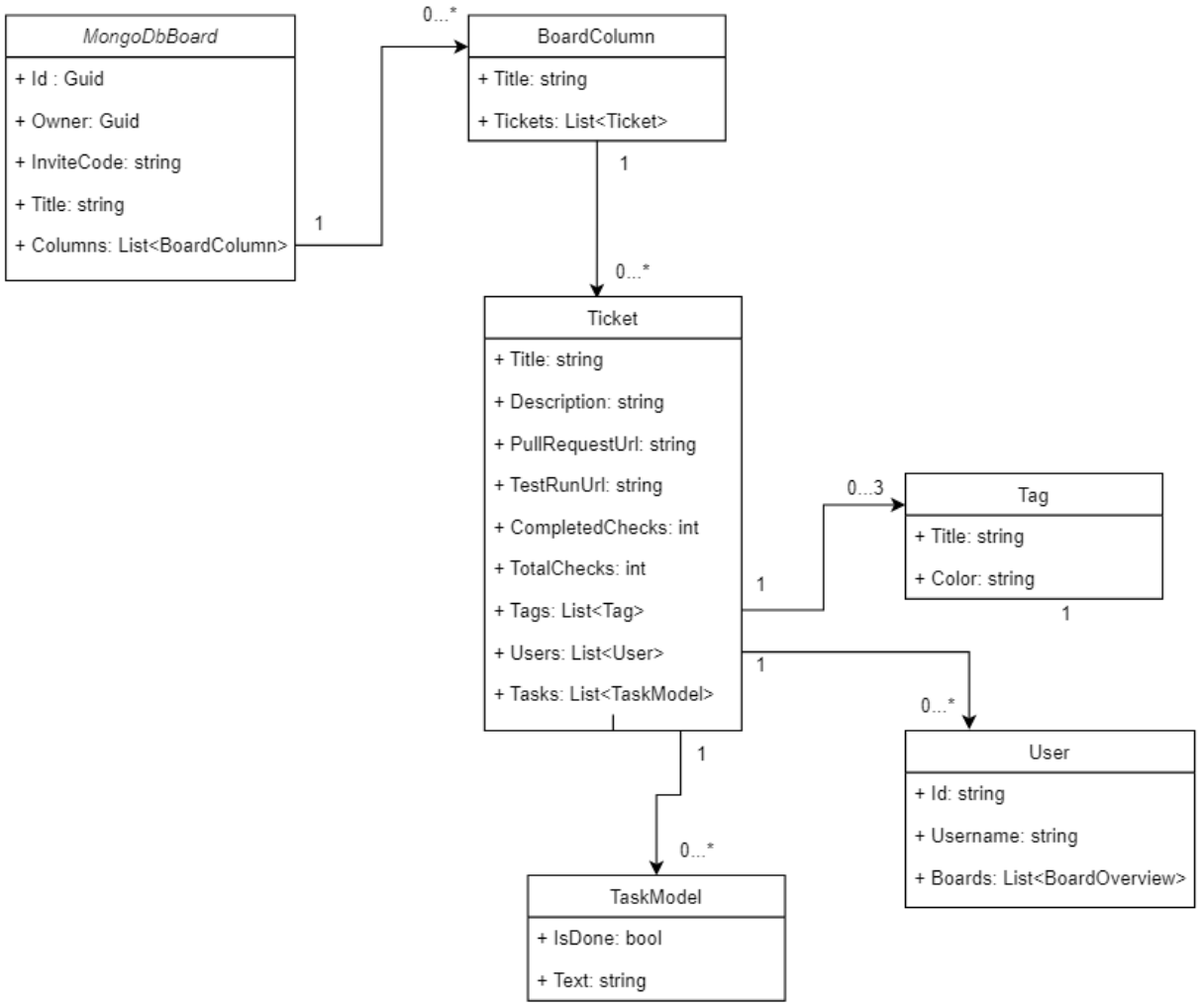

Abbildung 11 - MongoDbBoard Klassendiagram

Sequenz Diagramm:

Unsere Requests passieren folgendermassen.

Von dem Client aus wird ein Request an die API / das Backend gemacht. Im Backend wird der Aufruf in einem Controller abgefangen und an den Service weitergeleitet. Der Service verarbeitet den Request und erledigt das nötige. Danach werden, falls nötig, die Daten in der Datenbank angepasst gespeichert oder gelöscht. Dieser Request wird dann zurück an den Controller geleitet, welches eine Response erstellt. Dieser kann Daten, ein Success oder ein Fehler an den Client liefern. Der Client verarbeitet diese Infos und das UI wird entsprechend angepasst.

Hier ist ein Beispiel wie ein solcher Request aussehen kann.

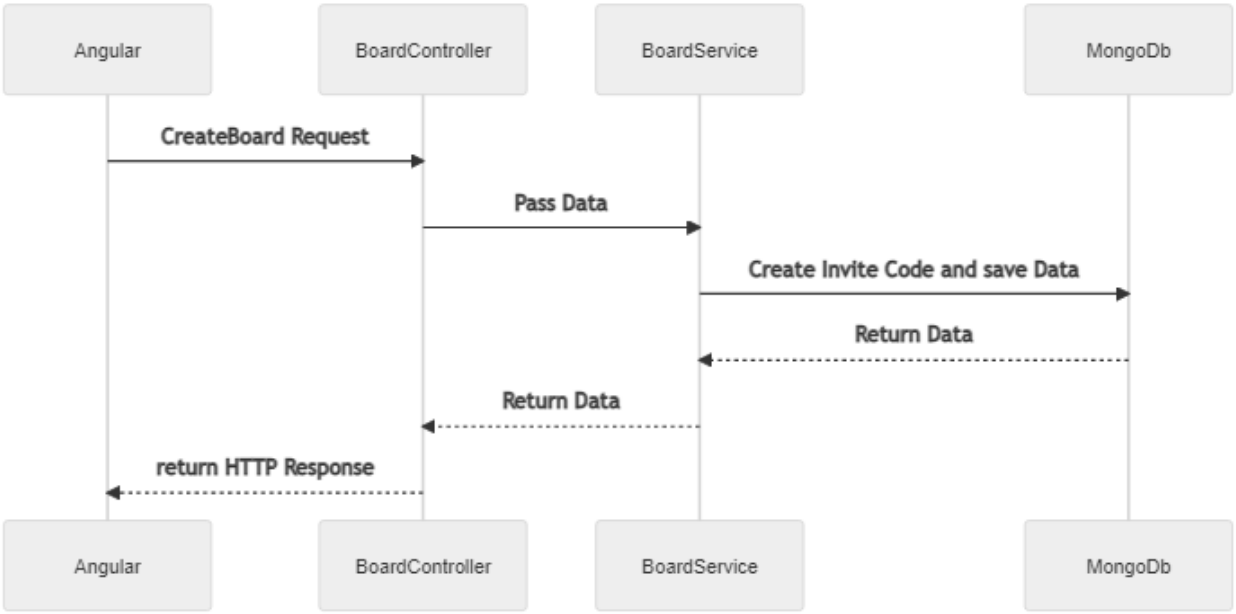

Abbildung 12 - Sequenzdiagram

4.6 Kontrollieren

In der Phase Kontrollieren der Projektmethode IPERKA geht es darum, das Projekt auf die vorausgesetzte und erwartete Funktionalität zu testen. Hier werden gefundene Fehler beschrieben und die komplette Applikation aufgrund des Testprotokolls getestet.

4.6.1 Manuelle Tests

Name	Geschäfts Notebook (NBDEV15)
Prozessor	Intel i7 Generation 8
Ram	32 GB
Browser	Chrome Browser Version:
Betriebssystem	Windows 10 Enterprise 64-Bit (Version: 22H2)
Grafikkarte	Intel UHD Graphics 620

Durchführung der Manuellen Tests

Protokoll: 1 – Testfall-Nr: 1	Registration eines neuen Benutzers

Erhaltenes Resultat	• Ich habe gab de URL: localhos:4200/register ein
	• Füllte das Formular wie folgt aus:
	  ○ Username: Andrin
	  ○ Password: AndrinTaskchain123!
	  ○ Confirm Password: AndrinTaskchain123!
	• Auf register geklickt
	• Ich wurde auf die dashboard Seite weitergeleitet und ich sehe nun die Optionen: «Join Board» und «+ Add Board» sowie in der Navbar sein username und die Optionen «Dashboard» und «Logout»

Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 2	Registration eines neuen Benutzers mit bereits vorhandenem username

Erhaltenes Resultat	• Ich öffnete die URL: localhost:4200/register
	• Ich habe das Formular wie folgt ausgefüllt:
	  ○ Username: Philipp
	  ○ Password: PhilippTaskchain123!
	  ○ Confirm Password: PhilippTaskchain123!
	• Ich klickte auf Register
	• Ich wurde auf die dashboard Seite weitergeleitet, auf der ich die Optionen sah: «Join Board» und «+ Add Board» sowie in der Navbar mein username und die Optionen «Dashboard» und «Logout»
	• Ich klickte auf Logout
	• Öffnete die register URL erneut
	• Füllte das Formular aus wie folgt:
	  ○ Username: Philipp
	  ○ Password: PhilippTaskchain123!
	  ○ Confirm Password: PhilippTaskchain123!
	• Ich klickte auf Register
	• Sobald ich den Benutzernamen eingegeben habe, tauchte eine rote Meldung auf, die mich darauf hinwies, dass der Benutzername bereits vorhanden ist.
	• Beim Klick auf register geschah nichts

Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 3	Registration eines neuen Benutzers mit keinem Benutzernamen

Erhaltenes Resultat	• Ich öffnete die URL: localhost:4300/register
	• Ich füllte das Formular aus wie folgt:
	  ○ Username: «leer»
	  ○ Passwort: KuSdi123!
	• Ich klickte auf register
	• Eine Meldung ist aufgetaucht in welcher stand, dass der Benutzername angegeben werden muss.

Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 4	Login eines Benutzers

Erhaltenes Resultat	• Ich öffnete die URL: localhost:4200/login
	• Ich füllte das Formular aus wie folgt:
	  ○ Username: Andrin
	  ○ Passwort: AndrinTaskchain123!
	• Ich klickte auf login
	• Ich wurde auf die dashboard Seite weitergeleitet auf der ich die Optionen «Join Board» und «+ Add Board» sah, sowie in der Navbar mein username und die Optionen «Dashboard» und «Logout».

Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 5	Login eines Benutzers mit falschen Anmeldedaten

Erhaltenes Resultat	• Ich öffnete die URL: localhost:4200/login
	• Ich füllte das Formular aus wie folgt:
	  ○ Username: Andrinn
	  ○ Passwort: AndrinTaskchain123!
	• Ich klickte auf «Log In»
	• Eine Meldung ist erschienen, die mich darauf hinwies, dass ich meine Anmeldedaten kontrollieren sollte.

Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 6	Login eines Benutzers mit fehlendem Username
Erhaltenes Resultat	• Ich öffnete die URL: localhost:4200/login • Ich habe in dem Formular nur das Passwort eingegeben und den Benutzername leer gelassen. • Ich habe auf login geklickt • Eine Meldung ist erschienen die mich darauf hingewiesen hat, dass der Benutzername angegeben werden muss.
Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 7	Ein Benutzer meldet sich an und navigiert auf eine Board Übersicht

Erhaltenes Resultat	• Ich öffnet die URL: localhost:4200/login
	• Ich füllte das Formular wie folgt aus:
	  ○ Username: Andrin
	  ○ Passwort: AndrinTaskchain123!
	• Ich klickte auf login
	• Ich wurde auf die dashboard Seite weitergeleitet
	• Auf der dashboard Seite klickte ich auf das erste Board
	• Ich wurde auf die Board Übersicht weitergeleitet und sah somit meine Spalten und Tickets, die ich bereits in diesem Board hatte.

Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 8	Ein angemeldeter Benutzer erstellt ein neues Board
Erhaltenes Resultat	• Ich öffnete die URL: localhost:4200/dashboard (ich habe mich bereits angemeldet) • Auf der Seite habe ich auf «+ Add Board» geklickt • In dem Dialog, der aufgetaucht ist, habe ich den Titel «Entwicklung» eingegeben • Ich klickte auf Save • Ich wurde auf die Board Übersicht weitergeleitet und könnte nun Spalten und Tickets erstellen. Ich sah der eingegebene Titel, sowie ein Settings Knopf daneben
Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 9	Ein angemeldeter Benutzer erstellt ein neues Board, ohne ein Titel anzugeben
Erhaltenes Resultat	• Ich öffnete die URL: localhost:4200/dashboard (ich habe mich bereits angemeldet) • Auf der Seite habe ich auf «+ Add Board» geklickt • In dem Dialog, der aufgetaucht ist, habe ich kein Titel eingegeben • Ich klickte auf Save • Es ist eine Meldung aufgetaucht, die mich darauf hingewiesen hat, dass ich ein Titel angeben muss.
Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 10	Ein angemeldeter Benutzer erstellt ein neues Ticket

Erhaltenes Resultat	• Ich hatte mich bereits angemeldet
	• Ich öffnete die Übersicht eines bestehenden Boards
	• In der ersten Spalte klickte ich unten auf +Add Card
	• Ich füllte das Ticket wie folgt aus:
	  ○ Titel: Benutzer Registration
	  ○ Im Editor: TODOS werden noch definiert.
	• Ich klickte auf «Save»
	• Der Ticket Dialog hat sich geschlossen und das erfasste Ticket erschien in der ersten Spalte

Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 11	Ein angemeldeter Benutzer editiert ein Ticket
Erhaltenes Resultat	• Ich hatte mich bereits angemeldet • Ich öffnete die Übersicht eines Bestehendes Boards welches auch schon erfasste Tickets hatte • Ich klickte auf ein bestehendes Ticket • Der Ticket Dialog hat sich geöffnet • Ich löschte den bestehenden Titel und ersetzte ihn mit «Benutzer Login» • Ich klickte auf save • Der Ticket Dialog hat sich geschlossen und der editierte Titel war ersichtlich in der Spalte bzw. auf dem Ticket.
Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 12	Ein angemeldeter Benutzer fügt sich selbst einem Ticket hinzu
Erhaltenes Resultat	• Ich hatte mich bereits angemeldet • Ich öffnete die Übersicht eines Bestehendes Boards welches auch schon erfasste Tickets hatte • Ich klickte auf ein bestehendes Ticket • Ich klickte auf Add bei der Benutzer Liste • Ich klickte auf den Hinzufügen Knopfe neben meinem Namen • Der Dialog mit der Benutzer Liste hat sich geschlossen und mein Name erschien auf dem Ticket in der Benutzer Liste.
Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 13	Ein angemeldeter Benutzer joined einem bestehenden Board mit einem Invite Code
Erhaltenes Resultat	• Ich hatte mich bereits angemeldet • Auf der dashboard Seite klickte ich auf «Join Board» • Ich wurde auf die /join Seite weitergeleitet • In dem Input Feld habe ich den Invite code SIOKHN eingegeben, welcher ich von einem bestehenden Board (einer anderen Person) kopierte • Ich wurde auf die Board Übersicht weitergeleitet
Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

Protokoll: 1 – Testfall-Nr: 14	Ein angemeldeter Benutzer gibt einen Invaliden Invite Code an
Erhaltenes Resultat	• Ich hatte mich bereits angemeldet • Auf der dashboard Seite klickte ich auf «Join Board» • Ich wurde auf die /join Seite weitergeleitet • In dem Input Feld habe ich den Invite code OKIHVB7TI angegeben • Eine rote Meldung ist unterhalb des Feldes aufgetaucht, welche beschrieb, dass der Invite Code invalid ist,
Status	Bestanden
Datum	24.03.2023
Getestet von	Dario Schaffner

5 Auswerten

Das Auswerten ist der letzte Schritt der IPERKA Projektplanungsmethode. In diesem Teil wird reflektiert, Erfolge und Misserfolge erkannt, sowie Verbesserungsmöglichkeiten gesucht und zum Schluss ein Fazit daraus gezogen.

5.1 Reflexion

Das Projekt hat im Grossen und Ganzen reibungslos geklappt, auf wirkliche Probleme bin ich zum Glück nicht gestossen. Ein grosser Teil hat sicherlich auch die ausführliche Planung dazu beigetragen, sowie auch die Gespräche mit dem Experten.

Ich habe durch die ganze Projektarbeit ein genaues Ziel, welches ich anstrebte und auch durchgezogen haben.

5.2 Erfolge

Mein grösster Erfolg ist, dass ich all meine Ziele, die ich mir vorgenommen habe, erreicht habe. Ich konnten alle geplanten Features umsetzen und testen. Zudem hatte ich noch genügend Zeit alle Features gründlich zu Testen und dementsprechend Bugs zu fixen. Ich konnte alle offenen Fragen stehts beantworten und falls ich Hilfe brauchte, war die Lehrperson oder der Experte stehts zur Hilfe da. Der detaillierte Zeitplan hat mir geholfen strukturiert zu meinem Ziel zu kommen. Dieser war oft sehr akkurat und eine gute Orientierung um weitere Schritte anzugehen.

5.3 Misserfolge

Misserfolge gab es nicht viele, da ich sehr fleissig gearbeitet, geplant und kommuniziert haben. Dennoch gab es Punkte, bei welchen Verbesserungspunkte vorhanden sind. Eigentlich hatte ich mir vorgenommen, den letzten Tag ausschliesslich für das Review der Arbeit zu reservieren. Da ich noch ein paar kleinere Pendenzen offen hatte, hat dies nicht ganz funktioniert. Ein weiterer Punkt, bei dem ich mich noch verbessern könnte, ist die Eintragung der Ist-Zeit im Zeitplan. Ich hätte aktiver und genauer schauen sollen, wie viel Zeit ich wirklich für welche Aufgabe brauchte. Manchmal kam es dazu das die Ist Zeit eher eine Schätzung nach Gefühl war weder eine ganz genaue Angabe.

Zum Schluss kommen wir noch zu einem Punkt, der meiner Meinung nach kein Misserfolg ist, jedoch das Erfüllen einer Projektarbeit erleichtern könnte. Mir ist aufgefallen das ich mir selber sehr viele Anforderungen gegeben habe. Ich habe die Zeit die wir zur Verfügung haben deutlich überschätzt und habe mir deshalb sehr viel vorgenommen. Dies hat zu einem immensen Stress geführt.

5.4 Verbesserungsmöglichkeiten

Aufgrund der Punkte, welche ich bei den Misserfolgen aufgelistet habe, habe ich nun eine Liste mit Verbesserungsmöglichkeiten erstellt.

- Projekt 1-2 Tage vor Schluss komplett abschliessen, damit die letzten Stunden ausschliesslich zum Reviewen und Überprüfen der Kriterienlisten zur Verfügung steht.
- Die Ist-Zeit im Zeitplan jedes Mal nach abschliessen einer geplanten Arbeit genauer eintragen.
- Die Zeit, die ich brauche für gewisse aufgaben genauer aufnehmen / aufschreiben damit ich nicht schätzen muss.
- Für die nächsten Projekte die Zeit besser einschätzen die wir insgesamt zur Verfügung haben.

5.5 Fazit

Ich bin ich äußerst zufrieden und stolz darauf, dass das selbstständige Projekt, welches ich in Angriff genommen habe, letztendlich ein voller Erfolg war. Obwohl es in der Schlussphase des Projekts zu einigen Stresssituationen gekommen ist und es vereinzelt zu Unklarheiten kam, konnten sämtliche funktionalen und nicht-funktionalen Anforderungen letztendlich erfolgreich erreicht und implementiert werden.

Es war eine anspruchsvolle Aufgabe, die ich allerdings durch meine Fähigkeit zur effektiven Organisation, Priorisierung und Problemlösung meistern konnte. Ich habe meine Verantwortlichkeiten stets im Blick behalten, um das bestmögliche Ergebnis sicherzustellen.

In diesem Projekt habe ich wertvolle Erfahrungen gesammelt, indem ich Herausforderungen bewältigt und effektive Strategien entwickelt habe, um auf unerwartete Probleme zu reagieren. Ich bin dankbar dafür, dass ich die Chance hatte, mein Engagement und meine Fähigkeiten als Projektleiter unter Beweis zu stellen.

Ich bin zu dem Schluss gekommen, dass ein erfolgreiches Projekt nicht nur von der Qualität der Arbeit abhängt, sondern auch von der Fähigkeit, sich auf veränderliche Bedingungen und unvorhergesehene Herausforderungen anzupassen. Es war eine wertvolle Erfahrung, die mich weiterhin bei zukünftigen Projekten und Aufgaben unterstützen wird.

6 Quellenverzeichnis

- Angular: https://angular.io/
- IPERKA: https://www.bexio.com/de-CH/blog/view/iperka-methode
- .Net Core (C#): https://learn.microsoft.com/en-us/dotnet/core/introduction
- Mongo DB: https://www.mongodb.com/home
- Sitemap : https://de.wikipedia.org/wiki/Sitemap
- Imgur: https://imgur.com/
- Angular Material: https://material.angular.io/
- Ngx-color: https://www.npmjs.com/package/ngx-color

7 Glossar

API -> Eine API (Application Programming Interface) ist eine Schnittstelle, die ermöglicht, dass verschiedene Computerprogramme miteinander kommunizieren und Daten und Funktionen austauschen können.

Client -> Der Client ist ein Gerät, welches mit einem Netzwerk verbunden ist. Er sendet Anfragen an den Server und der Server liefert die gewünschten Daten oder Dienste zurück.

Sitemap -> Die Sitemap soll beim Realisieren einen guten Überblick über die verschiedenen Seiten und deren Relation verschaffen. Sie zeigt somit die logische Struktur der Webapplikation auf.

8 Abbildungsverzeichnis

9 Anhang

9.1 Quellcode

Backend:

UserController:

```
using api.Models;
using api.Models.request;
using api.Models.response;
using Microsoft.AspNetCore.Mvc;

namespace api.Controllers;

[ApiController]
[Route("api/[controller]")]
public class UserController : BaseController
{
    [HttpPost]
    [Route("register")]
    public IActionResult Register(UserRegister userRegister)
    {
        User user = UserService.RegisterUser(userRegister);

        if (user == null)
        {
            return BadRequest ();
        }

        return Ok(user);
    }

    [HttpPost]
    [Route("login")]
    public IActionResult Login (UserLogin user)
    {
        try
        {
            var result = UserService.LoginUser(user);

            if (result == null)
            {
                return NotFound ();
            }

            return Ok(result);
        }
        catch
        {
            return NotFound ();
        }
    }

    [HttpGet]
    [Route("usernames")]
    public IActionResult GetTakenUsernames ()
    {
        List<string> usernames = UserService.GetTakenUsernames();

        return Ok(usernames);
```

```
    }

    [HttpPost]
    [Route("join")]
    public IActionResult JoinBoard (JoinBoardRequest request)
    {
        Board foundBoard = BoardService.GetBoardByInviteCode(request.InviteCode);

        if (foundBoard != null && !request.User.Boards.Any(board => board.Id ==
foundBoard.Id))
        {
            UserService.AddUserToBoard(request. User, foundBoard);

            JoinBoardResponse response = new ()
            {
                BoardId = foundBoard.Id,
                User = request. User
            };

            return Ok(response);
        }

        return NotFound();
    }

    [HttpPost]
    [Route("boards")]
    public IActionResult GetBoards(GetBoardsRequest request)
    {
        List<BoardOverview> boards = UserService.GetBoards(request.User);

        GetBoardsResponse response = new()
        {
            Boards = boards
        };

        return Ok(response);
    }

    [HttpPost]
    [Route("users")]
    public IActionResult GetAllUsers(GetAllUsersRequest request)
    {
        List<User> users = UserService.GetAllUsers(request.BoardId);

        GetAllUsersResponse response = new()
        {
            Users = users,
        };

        return Ok(response);
    }

    [HttpPost]
    [Route("removeUser")]
    public IActionResult RemoveUserFromBoard(RemoveUserRequest request)
    {
        UserService.RemoveUserFromBoard(new Guid(request.UserId), new
Guid(request.BoardId));

        return Ok();
    }
```

```
}
```

BoardController:

```
using api.Models;
using api.Models.request;
using api.Models.response;
using Microsoft.AspNetCore.Mvc;

namespace api.Controllers;

[ApiController]
[Route("api/[controller]")]
public class BoardController : BaseController
{
    [HttpPost]
    [Route("createBoard")]
    public IActionResult CreateBoard(CreateBoardRequest request)
    {
        Board board = BoardService.CreateBoard(request.BoardTitle, request.User);

        if (board == null)
        {
            return BadRequest();
        }

        UserService.AddUserToBoard(request.User, board);

        CreateBoardResponse response = new()
        {
            Board = board
        };

        return Ok(response);
    }

    [HttpPost]
    [Route("save")]
    public IActionResult SaveBoard(SaveBoardRequest request)
    {
        Board board = BoardService.SaveBoard(request.Board);

        SaveBoardResponse response = new()
        {
            Board = board
        };

        return Ok(response);
    }

    [HttpPost]
    [Route("board")]
    public IActionResult GetBoard(GetBoardRequest request)
    {
        Board board = BoardService.GetBoardById(new Guid(request.BoardId));

        return Ok(board);
    }

    [HttpPost]
    [Route("deleteBoard")]
    public IActionResult DeleteBoard(DeleteBoardRequest request)
    {
```

```
        BoardService.DeleteBoard(new Guid(request.BoardId));

        foreach (User user in request.Users)
        {
            UserService.RemoveUserFromBoard(new Guid(user.Id), new
Guid(request.BoardId));
        }

        return Ok();
    }
}
```

Frontend:

Board Service:

```
import { HttpClient } from '@angular/common/http';

import { Injectable } from '@angular/core';

import { Observable } from 'rxjs';

import { BoardModel } from '../models';

import {

 ICreateBoardRequest,

 IDeleteBoardRequest,

 IGetBoardRequest,

 ISaveBoardRequest,

} from '../models/request';

import { ICreateBoardResponse, ISaveBoardResponse } from '../models/response';


@Injectable({

 providedIn: 'root',

})

export class BoardService {

 private baseurl = 'https://localhost:7079/api/Board';


 constructor(private http: HttpClient) {}


 /**

  * API call to load a board
```

```
  * @param {IGetBoardRequest} request
  * @return {*}  {Observable<BoardModel>}
  * @memberof BoardService
  */
 getBoard(request: IGetBoardRequest): Observable<BoardModel> {
   const url = this.baseurl + '/board';
   return this.http.post<BoardModel>(url, request);
 }

 /**
  * API call to save a board
  * @param {ISaveBoardRequest} request
  * @return {*}  {Observable<ISaveBoardResponse>}
  * @memberof BoardService
  */
 saveBoard(request: ISaveBoardRequest): Observable<ISaveBoardResponse> {
   const url = this.baseurl + '/save';
   return this.http.post<ISaveBoardResponse>(url, request);
 }

 /**
  * API call to create a new board
  * @param {ICreateBoardRequest} request
  * @return {*}  {Observable<ICreateBoardResponse>}
  * @memberof BoardService
  */
 createBoard(request: ICreateBoardRequest): Observable<ICreateBoardResponse> {
   const url = this.baseurl + '/createBoard';
   return this.http.post<ICreateBoardResponse>(url, request);
 }
```

```
  /**
   * API call to delete a board
   * @param {IDeleteBoardRequest} request
   * @return {*}  {Observable<void>}
   * @memberof BoardService
   */
  deleteBoard(request: IDeleteBoardRequest): Observable<void> {
    const url = this.baseurl + '/deleteBoard';
    return this.http.post<void>(url, request);
  }
}
```