Bachelorarbeit

2PLAN

Ostschweizer Fachhochschule

Abteilung Informatik

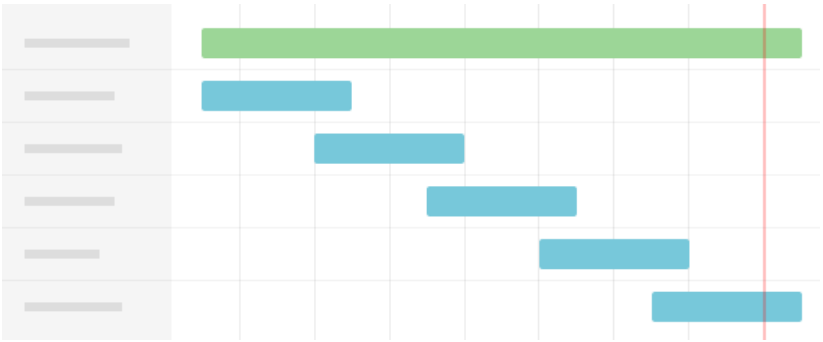
Frühlingssemester 2021

Autoren	Julia Tanner	**Betreuer**	Silvan Gehrig
	Severin Amacher	**Experte**	Felix Egli
		Auftraggeber	Raphael Ritter, 2BIT GmbH
		Gegenleser	Cyrill Brunschwiler

OST
Ostschweizer
Fachhochschule

Bachelorarbeit 2PLAN

Thema: Angular Component Library



Autor:	Silvan Gehrig
Version:	1.0
Erstellt am:	13.09.2020
Letzte Änderung am:	09.02.2021

1. Einführung

Planung von Projekt-Ressourcen ist ein sehr anspruchsvolles Thema. Im Rahmen dieser Bachelorarbeit soll ein benutzerfreundliches Planungstool als Angular Component Library mit dem Namen *2PLAN* erarbeitet werden.

Die Planungsdaten werden von mehreren Microservices zur Verfügung gestellt.

2. Aufgabe

Das Ziel der Arbeit besteht darin, eine Angular Library zur Ressourcenplanung inklusive Demo Applikation zu erstellen.

Die Ressourcenplanung kann verschiedenste Ausprägungen haben, die alle damit abgedeckt werden sollen. Folgende Einsatzzwecke sind denkbar (Liste ist nicht abschliessend):

- Projektplanung
- Schichtenplanung
- Einsatzplanung

Potenzielle Konflikte als auch fehlende Ressourcen werden einfach und übersichtlich dargestellt. Diese Konflikte / fehlende Ressourcen können einfach aufgelöst werden. Es kann im Planner zwischen einer Gesamt (Unternehmung)-, ein Vorhaben- als auch Ressourcenansicht hin und her gewechselt werden.

Geplant wird jeweils auf Wochenbasis (optional mit Wochenende), wobei auch eine Tages- / Monatsansicht vorhanden ist. Pro Ressource kann ein Arbeitsplan (Woche, Monat) ausgedruckt werden.

Die Software unterstützt die folgenden Benutzerrollen:

- Admin
- Planer
- User

Hauptziel des Produktes ist es, die Ressourcenplanung so einfach wie möglich auszugestalten. Dabei stehen Bedienbarkeit, Übersichtlichkeit und Effizienz im Vordergrund. Der Fokus liegt klar auf einer sehr guten User Experience.

Die Aufgabenstellung kann bei Bedarf mutiert werden.

Die Arbeit besteht aus den folgenden Hauptaufgaben:

2.1 Requirements Analysis

- Ausarbeiten der Anforderungen an die zu erarbeitende Library in Zusammenarbeit mit dem Auftraggeber.
- User Centered Design bei der Ausarbeitung der Ressourcenplanungs-Komponente(n).

2.2 Frontend

- Erstellen der Funktionalität als Angular Library und einbinden in eine Angular Demo Applikation.
- Optionale Erweiterungen
  - Planungsalgorithmus
  - OAuth Integration

2.3 Abgrenzungen

- Das .NET Core Backend sowie die dafür benötigte Entwicklungsumgebung wird vom Auftraggeber zur Verfügung gestellt.

3. Themen und Technologien

- Frontend
  - Angular mit Setup des Industriepartners, Material Design
- Backend
  - .NET Core

4. Erwartete Resultate

Als Resultat soll eine dokumentierte und in der Kundenumgebung lauffähige Applikation abgegeben werden. Eine gute Wartbarkeit der Software soll die Weiterentwicklung sicherstellen. Weiteres gilt zu beachten:

- Vorgehen nach den Regeln des Software-Engineering (Scrum und UP), Arbeit nach Projektplan. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten.
- Erfassen des tatsächlichen Arbeitsaufwands. Es muss ersichtlich sein, wer für welchen Teil der Arbeit und des Berichts verantwortlich ist.
- Alle Dokumente sind nachzuführen, d.h. sie sollen den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren.

Abstract

Die Planung von Projekt-Ressourcen ist ein sehr anspruchsvolles und zeitaufwändiges Thema, das verschiedenste Branchen betrifft. Von der Firma 2BIT GmbH haben wir den Auftrag erhalten, eine übersetzbare Angular Component Library zu entwickeln, die den gesamten Planungsprozess abdeckt. Das Hauptziel ist es, die Ressourcenplanung so einfach wie möglich auszugestalten. Dabei steht eine gute User Experience, die Bedienbarkeit und die Übersichtlichkeit im Vordergrund.

Als Erstes wurde eine sorgfältige Anforderungsanalyse durchgeführt, um die Anforderungen an das Produktes zu verstehen. Anschliessend wurden klickbare Mockups erstellt und mit dem Auftraggeber analysiert. Gegen Ende der Implementierungsphase wurde das Produkt mit einer User Experience Designerin geprüft und verbessert, um eine möglichst hohe User Experience zu erreichen.

Die Angular Component Library 2PLAN stellt alle Komponenten zur Verfügung, die für einen Planungsprozess benötigt werden. Um die Planung durchführen zu können, enthält die Bibliothek eine ressourcen- und eine projektbasierte Gantt-Chart Komponente. Falls bei der Planung Konflikte entstehen, hat der Planer die Möglichkeit, den Konflikt zu akzeptieren oder aufzulösen. Ausserdem enthält die Library eine Dienstplan-Komponente. Mit dieser Komponente kann eine Ressource ihren Arbeitsplan einsehen und Arbeiten als abgeschlossen markieren. Für alle Komponenten wird eine Wochen- und eine Monatsansicht zur Verfügung gestellt.

Zusätzlich zu 2PLAN wurde eine Demo-Applikation entwickelt. Diese bietet eine Stammdatenverwaltung an, um die benötigten Daten für eine Planung zu erfassen und bindet die Komponenten von 2PLAN ein.

Management Summary

Ausgangslage

Da KMU typischerweise keine oder nur eine manuelle Planung haben, hat uns die Firma 2BIT GmbH damit beauftragt, ein Planungsinstrument für KMU zu entwickeln. Wichtig ist, dass das Produkt nicht nur für eine Projektplanung, sondern auch für andere Ausprägungen wie zum Beispiel für eine Schichtplanung entwickelt werden soll.
Das Planungsinstrument soll in Form einer Angular Component Library entwickelt und schlussendlich in einer Demo-Applikation eingebunden und demonstriert werden.

Vorgehen / Technologie

Zu Beginn der Arbeit haben wir vom Auftraggeber das Grundgerüst der Demo-Applikation mit einem Angular Material Design Setup erhalten. Ausserdem wurde uns ein .NET Core Backend mit einer Login-Funktion aufgesetzt, sodass wir uns hauptsächlich auf das Frontend konzentrieren konnten.
Über die ganze Arbeit hinweg bestand eine enge Zusammenarbeit mit Mitarbeitern von 2BIT GmbH und Silvan Gehrig, um Anforderungen zu analysieren, Resultate zu reflektieren und Fragen zu klären.

Ergebnisse

Das Ergebnis unserer Bachelorarbeit ist eine mit Angular 11 entwickelte und übersetzbare Library namens 2PLAN, die als NPM Package installiert werden kann.

2PLAN bietet verschiedene Komponenten an, die einen beim Planungsprozess unterstützen sollen:

- Für die Planung steht eine ressourcen- und eine projektbasierte Gantt-Komponente zur Verfügung. Mithilfe dieser Komponente kann eine Arbeitszuteilung durchgeführt werden. Da es bei einer Planung immer wieder zu Konflikten kommen kann, wurde im .NET Core Backend der Demo-Applikation ein Regelwerk hinterlegt. Falls eine dieser Regeln fehlschlägt, wird ein Konflikt erstellt und in den Gantt-Komponenten farblich hervorgehoben. Zusätzlich dazu gibt es eine Komponente für die Meldungen, damit ein Planer diese akzeptieren oder auflösen kann.
- Ein weiterer, grosser Bestandteil von 2PLAN ist die Dienstplan-Komponente. Mithilfe dieser Komponente können die Ressourcen ihren Arbeitsplan auf Wochen- oder Monatsbasis einsehen. Ausserdem können sie eine Arbeit als erledigt markieren, sobald sie diese abgeschlossen haben.

Ausblick

Der Standpunkt unserer Bibliothek bietet einen guten Ansatz, um eine Planung durchführen zu können. Dennoch gibt es einige Erweiterungen, die noch eingebaut werden könnten, um die Planung zu vereinfachen. Eine Idee wäre zum Beispiel, dass im Dienstplan und in der Gantt-Komponente der Monat über einen Swipe gewechselt werden kann. Ausserdem wäre Drag and Drop ein sinnvolles Feature für die Planung, damit Allokationen ohne den Dialog verschoben werden können. Schliesslich wäre eine automatisierte Planung mittels Planungsalgorithmus eine sehr sinnvolle Erweiterung. Dieser könnte die bestmögliche Planung erstellen, die bei Bedarf vom Planer modifiziert werden kann.

Danksagungen

Wir danken den folgenden Personen für ihre Unterstützung während unserer Bachelorarbeit:

- Institut für Software, Silvan Gehrig
- 2BIT GmbH, Raphael Ritter
- 2BIT GmbH, Maxim Kuzmin
- Abacus Research AG, Priska Steiger (User Experience Designerin)
- Jeannine Tanner (Lektorat Anwenderdokumentation)
- Martin Lengwiler (Lektorat)

Inhaltsverzeichnis

Glossar und Abkürzungsverzeichnis

BA Abkürzung für Bachelorarbeit. S. 5, 52

DOM Abkürzung für Document Object Model. S. 44, 48

KMU Abkürzung für kleine und mittlere Unternehmen. S. v, 7

Logische Code-Zeilen Code-Zeilen, die interpretiert wurden. S. 50

MIT License Die MIT License ist eine Open Source Lizenz. Sie erlaubt das Wiederverwenden der Software für Open Source Projekte oder auch für Software, deren Quelltext nicht frei einsehbar ist. S. 21–23

MVP Abkürzung für Minimum Viable Product. S. 54, 58

NPM NPM ist ein Paketmanager, der es ermöglicht, Packages zu teilen oder private Packages zu verwalten. S. v, 2, 3, 30, 31

Open Source Software mit öffentlichem Quellcode, der von Dritten eingesehen, geändert und genutzt werden kann. Meistens sind Open Source Software kostenlos. S. 21–23

Overhead Überschuss an Rechenzeit, Speicher, Bandbreite oder anderen Ressourcen. S. 39

Physische Code-Zeilen Code-Zeilen, die in einer Datei stehen. S. 50

TMetric TMetric ist eine Time Tracking App. S. 56

UI Abkürzung für User Interface. S. 3, 32

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Technischer Bericht

Kapitel 1

Technischer Bericht

1.1 Ausgangslage

Die Planung von Projekt-Ressourcen ist ein anspruchsvolles Verfahren. Im Planungsprozess muss ein Planer den Überblick behalten, wann etwas abläuft, wer dies erledigt, was dafür benötigt wird usw. Wenn dem Planer ein Fehler unterläuft, kann dies verheerende Auswirkungen für den Projektverlauf und schlussendlich auf den Erfolg des gesamten Projektes haben. Die Berücksichtigung all dieser Faktoren wird zunehmend schwieriger, je grösser eine Firma ist. Wenn viele Projekte gleichzeitig laufen, werden vielleicht Ressourcen doppelt gebucht oder nicht komplett ausgelastet. Um dies zu verhindern und um den Planer bei seiner Arbeit zu unterstützen, hatte 2BIT die Idee für das 2PLAN Projekt.

2PLAN soll ein benutzerfreundliches Planungstool in Form einer Angular Component Library abbilden. Der Planer soll in der Planungsart nicht eingeschränkt werden, sodass er je nach Situation eine Projektplanung, Schichtplanung, Einsatzplanung oder ähnliches umsetzen kann. Damit der Planer allfällige Konflikte im Auge behalten kann, sollen ihm diese auf der Planungsansicht angezeigt werden. Jedoch soll der Planer dabei nicht in der Ausführung seiner Arbeit behindert werden. Es soll ihm also ermöglicht werden, dass er zuerst einen Entwurf der Planung erstellen kann und sich erst später um die Konflikte kümmern kann. Der Planer soll ausserdem die Möglichkeit haben in der Planungsansicht zwischen einer Wochen- und Monatsansicht zu wechseln, damit er den Informationsfluss selbstständig einschränken oder erweitern kann.

Damit auch die Ressourcen über ihre Zuweisungen informiert sind, soll 2PLAN ausserdem über einen Arbeitsplan verfügen. Im Arbeitsplan können die Ressourcen ihre persönlichen Zuweisungen auf Wochen- oder Monatsbasis einsehen und den Planer über erledigte Arbeiten informieren.

1.2 Lösungskonzept

Um die Anwendung der Library zu demonstrieren, soll diese in einer Demo-Applikation eingebunden werden. In der Demo-Applikation soll man ausserdem die Möglichkeit haben, Stammdaten zu erfassen. Dafür greift die Applikation auf die API eines .NET Core Backends zu, das von 2BIT implementiert wurde und für das Datenmanagement verantwortlich ist. Das API wurde von uns nach einer Flow-Architektur definiert. Das bedeutet, dass die gesamte Applikation in Flows aufgeteilt wird. Immer wenn ein Flow gestartet wird, zum Beispiel bei der Projektplanung, werden alle Daten, die für den Flow benötigt werden, geladen.

Die 2PLAN Bibliothek wurde nicht bereits zu Beginn als NPM [21] Package implementiert, sondern zuerst als lokale Library [20]. Dies hatte den Vorteil, dass bei Änderungen die Library

nicht immer neu auf NPM veröffentlicht werden musste, um Änderungen in der Applikation zu sehen, sondern einfach lokal gebuildet werden konnte.

1.3 Umsetzung

Bei der Umsetzung gingen wir nach dem Angular Lehrbuch vor. Alle Workspaces, Projekte und Komponenten wurden über die Angular CLI [4] generiert. Die Codequalität wurde mit ESLint [12] und Codereviews sichergestellt. Für die Versionsverwaltung erstellte uns 2BIT ein Azure DevOps Repository. Bevor wir mit der Implementierung starteten, haben wir die vorgefertigte Pipeline noch ergänzt mit dem Builden und Testen der Library. Um sicher zu stellen, dass unsere Architektur auch funktioniert, implementierten wir zuerst einen Prototypen. Mit dem Prototypen wurden folgende Aspekte zur Überprüfung implementiert:

1. Eine Library mit ersten simplen UI Komponenten, die in einer Applikation eingebunden wurden, um den lokalen Library Ansatz zu testen.
2. Übersetzung mit ngx-translate [29] in der Library, um die Integration mit einer Library zu testen.
3. Beispiel Request an das Backend schicken, um die Verbindung zu testen.
4. Angular Material Design einsetzen, damit wir uns damit auseinandergesetzt haben.
5. Ein E2E Test mit Protractor [24] implementieren, um die Technologie kennen zu lernen.

Nachdem wir unseren Projektplan zusammen mit dem Prototypen dem Kunden präsentiert hatten, starteten wir mit der Implementation von 2PLAN. Am Anfang haben wir in den Komponenten mit hardcodierten Beispieldaten gearbeitet, da das API noch nicht bereit war, um Daten abzufragen. Da es in diesem Projekt wichtig war, ein gutes UI zu entwickeln, um eine hohe User Experience zu erreichen, fokussierten wir uns zuerst auf die Darstellung und das Design der Views und erst danach auf die Logik dahinter.

Zwei Wochen vor End of Construction legten wir den Feature Freeze an, damit wir einen Sprint lang Zeit hatten, um allfällige Bugs und Unschönheiten zu korrigieren. Somit konnten wir sicherstellen, dass wir das Projekt in einem möglichst sauberen Zustand an 2BIT abliefern können.

1.4 Ergebnisse

Als Endprodukt unserer Bachelorarbeit kam die gewünschte 2PLAN Library, sowie eine Web-Applikation, welche die Library einbindet, zustande. Die Web-Applikation ist an das Backend angebunden und stellt eine Übersicht für das Erstellen, Bearbeiten und Löschen sämtlicher Stammdaten zur Verfügung. Über die Library kann eine Planung nach verschiedenen Verfahren durchgeführt werden und die Ressourcen können ihren Dienstplan einsehen. Weiterhin wurden alle Views responsive implementiert, sodass die Applikation auf einem Mobile, Tablet oder einem Desktop verwendet werden kann. Da der Informationsgehalt bei der Planung sehr hoch ist, ist für diese View nicht vorgesehen, dass sie auf dem Mobile verwendet wird. Die View würde zwar funktionieren, jedoch stehen nicht alle Funktionen zur Verfügung.

1.5 Auswertung

Obwohl unsere Bachelorarbeit für uns sehr anspruchsvoll war, konnten wir 2BIT eine saubere und funktionierende Angular Library abliefern. Dies verdanken wir einerseits einer guten Planung von unserer Seite, mit der wir Probleme früh erkennen und gut daran vorbei navigieren konnten. Andererseits wurde gut mit unserem Kunden kommuniziert, damit wir seine

Wünsche genau verstehen und den besten Weg ausarbeiten konnten.
Leider gab es aber auch bei diesem Projekt einige Probleme. Obwohl wir grosses Mass an Ausdauer, Enthusiasmus und Neugierde zeigten, hatten wir einige Schwierigkeiten in Bezug auf die vielen, uns unbekannten Technologien.

1.5.1 Probleme

Angular

Mit fast keinen Vorkenntnissen in Angular war die Aufgabe am Anfang etwas überwältigend. Durch den Prototyp und Tutorials konnten wir uns etwas Wissen aneignen und grössere Probleme verhindern, jedoch benötigten wir oftmals mehr Zeit, weil wir zuerst alles recherchieren und verschiedene Konzepte erlernen mussten. Da uns dies aber bereits von Anfang an bewusst war, rechneten wir in der Planung immer genügend Zeit und Reserven ein.

E2E Tests

Auch bei den E2E Tests tauchten Probleme mit dem Protractor Framework auf. Obwohl wir uns damit bereits während der Prototyp-Phase auseinandergesetzt hatten, war die Implementation schlussendlich doch ziemlich mühselig. Dies lag daran, dass uns nicht bewusst war, was alles im Hintergrund abläuft. Zum Beispiel hatten wir ein Problem mit einer Timeout Exception, die auf einer bestimmten Seite auftrat, deren Ursprung aber unklar war. Nach langer Nachforschung fanden wir heraus, dass Protractor auf das Synchronisieren der View wartet und in dieser View ein setTimeout() implementiert wurde. Dies hatte zur Folge, dass Protractor auf das Timeout wartete und schlussendlich selber in einen Timeout lief.

API-Architektur

Das grösste Problem, das uns begegnete, betraf die API-Architektur. Das API wurde zwar von 2BIT implementiert, jedoch wurde die Definition von uns festgelegt. Dadurch konnten wir das API ohne längere Kommunikationsprozesse so gestalten, wie wir es in der Applikation benötigten. Da wir möglichst wenige Server Requests haben wollten, tendierten wir dazu das API mit der Graph-Architektur aufzubauen. Wir holten uns die Meinung unseres Betreuers ein; auch er war der Ansicht, dass dies Best Practice wäre. Als wir die API nach Graph-Architektur definiert hatten, präsentierten wir dies unserem Kunden. Da 2BIT tendenziell eher die Flow-Architektur bei API verwendet, tendierte er eher zu dieser Lösung. Ausserdem machte er uns darauf aufmerksam, dass wir mit der Graph-Architektur ein State-Management benötigen würden und dies eine weitere, für uns unbekannte Technologie bedeuten würde. Daraufhin wurde ein Meeting mit dem Betreuer und dem Kunden arrangiert, in dem wir uns für die Flow-Architektur entschieden, um möglichst nach Kunden Architektur vorzugehen und um kein weiteres Risiko mit einer unbekannten Technologie einzugehen. Dies führte zu Verzögerungen, da wir die API neu definieren mussten und diese noch gemäss unserer Definition implementiert werden musste.

1.5.2 Offene Punkte

Durch die zuvor beschriebenen Probleme verloren wir viel Zeit und erkannten schnell, dass wir nicht alle geplanten Use Cases (siehe Kapitel 2.2) implementieren werden können. Aus diesem Grund priorisierten wir die Use Cases, die zu diesem Zeitpunkt noch nicht implementiert wurden, mit unserem Kunden. Dabei legten wir fest, dass die Use Cases 5.1 und 7 (siehe Kapitel 2.2.6 und 2.2.8) sowie die Autorisierung der verschiedenen Rollen (siehe Kapitel 2.1) niedrigere Priorität haben und bei Zeitmangel auch weggelassen werden können. Schlus-

sendlich mussten wir diese drei Punkte auch effektiv weglassen, da uns die Zeit nicht mehr reichte.

1.5.3 Ausblick

In unserer BA konnten wir ein gutes Fundament implementieren, das noch nicht ganz für den produktiven Gebrauch geeignet ist, aber gut von 2BIT übernommen und fertig gestellt werden kann.
Um die Funktionalität der Library zu vervollständigen, wäre es bestimmt sinnvoll, die Features zu implementieren, für die wir nicht mehr genug Zeit hatten. Des Weiteren gibt es noch einige Performance-Optimierungen, welche die User Experience erhöhen würden. Dafür haben wir bereits eine Auswertung erstellt und mögliche Lösungsansätze dokumentiert (siehe Kapitel 8.3). Nach der Implementierung dieser zwei Punkte wäre 2PLAN ein stabiles Produkt, das Endkunden präsentiert werden könnte.

In Zukunft wären aber noch weitere Erweiterungen vorstellbar, welche die Usability des Produktes und die Planung vereinfachen würden. Zum Beispiel könnten an diversen Orten Swipe und Drag and Drop Funktionalitäten eingebaut werden. Dies würde die Arbeit des Planers, gerade auf mobilen Geräten, extrem vereinfachen. Des Weiteren könnte die Planung durch eine Smart Assistant erweitert werden. Dieser könnte zum Beispiel für eine Allokation bereits eine geeignete Ressource vorschlagen. So müsste der Planer die Informationen nicht mehr selber auswerten. Es wäre aber auch denkbar, dass es einen Planungsalgorithmus gibt, der die bestmögliche Planung ausrechnet und erstellt, sodass der Planer eine Basis hätte und dies später noch modifizieren und erweitern könnte.

Teil II

Projektdokumentation

Kapitel 2

Anforderungsspezifikation

Das Hauptziel von 2PLAN ist ein Planungsinstrument für KMU zu entwickeln. Die Ressourcenplanung soll dabei so einfach wie möglich gestaltet werden. Die Bedienbarkeit, Übersichtlichkeit und Effizienz stehen im Vordergrund.

2.1 Benutzerrollen

Die Informationen der Benutzerrollen werden aus dem Dokument "Anforderungen 2BIT" [25] von Raphael Ritter entnommen.

2.1.1 Admin

Der Admin ist zuständig für die Stammdatenverwaltung. Dazu gehört die Verwaltung der Vorhaben, Vorgänge, Ressourcen und Eigenschaften. Die Vorhaben kann er einem Planer für die Planung zuweisen.

Zusätzlich dazu kann der Admin alles, was der Planer auch kann. Einziger Unterschied ist, dass er alle Vorhaben sieht und nicht nur die, die ihm zugewiesen sind.

2.1.2 Planer

Der Planer kann alle ihm zugewiesenen Vorhaben planen und neue Vorgänge für ein Vorhaben erstellen. Die Planung kann er auf einer Gesamt-, einer Vorhabens oder einer Ressourcenansicht ausführen. Geplant wird auf Wochenbasis, wobei auch eine Tages- / Monatsansicht vorhanden ist. Fehlermeldungen und Warnungen können durch den Planer akzeptiert werden.

Der Planer kann einen Projektplan von einem Vorhaben als PDF exportieren.

2.1.3 User (Ressource)

Der User sieht die für ihn geplanten Vorhaben in einer Kalenderansicht. Ausserdem kann er seinen Einsatzplan als PDF ausdrucken lassen.

2.2 Use Cases

Die Use Cases wurden anhand der Meilensteine Core Construction, Alpha Release und Beta Release aufgeteilt und farblich markiert. Ausserdem ist erkennbar, welche Use Cases wir nicht mehr umsetzen konnten.

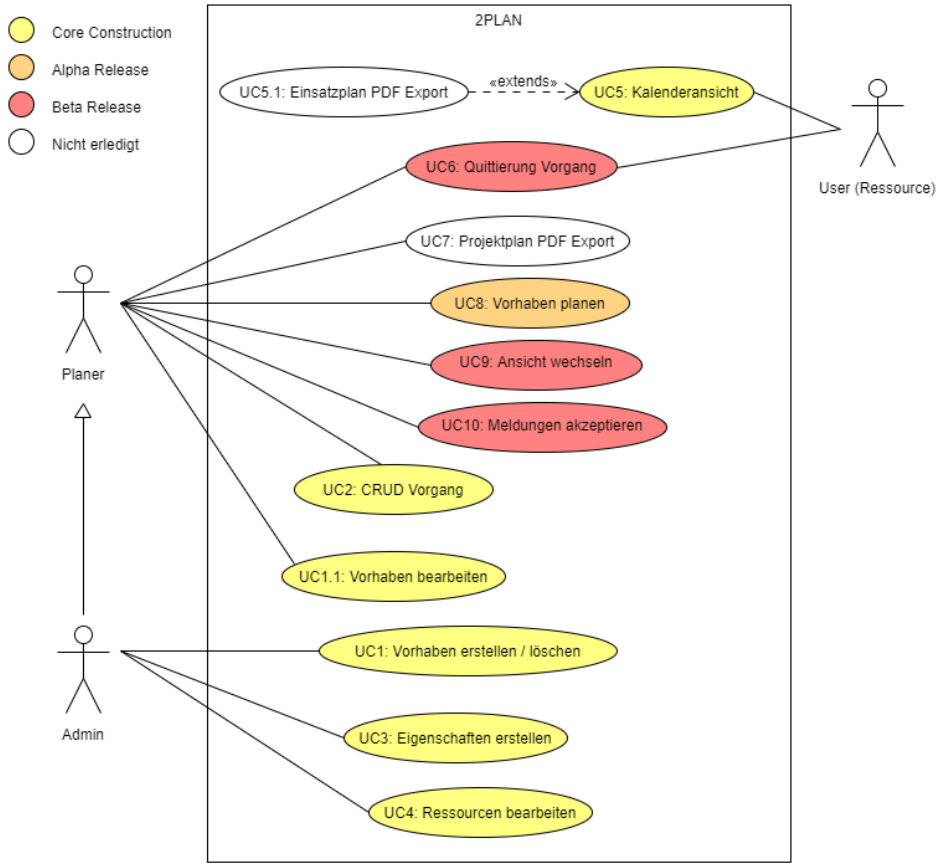


Abbildung 2.1: Use Case Diagramm

Nachfolgend wird jeder Use Case im Brief Format beschrieben.

2.2.1 UC1: Vorhaben erstellen / löschen

Main Actor	Admin
Main Success Scenario	Der Admin kann Vorhaben erstellen und löschen. Ein Vorhaben enthält ein Start- / Enddatum, einen Zustand (Entwurf, Aktiv, Abgeschlossen) und mindestens einen Vorgang. Ausserdem kann optional ein Zeitbudget definiert werden, das dann mit den effektiv verbuchten Stunden verglichen wird.

Tabelle 2.1: UC1

2.2.2 UC1.1: Vorhaben bearbeiten

Main Actor	Admin & Planer
Main Success Scenario	Der Admin / Planer kann ein Vorhaben bearbeiten. Das Start- / Enddatum darf nur soweit geändert werden, dass alle Vorgänge noch innerhalb von diesem Zeitraum liegen. Nach einer Änderung wird die optionale Validierung durchgeführt.

Tabelle 2.2: UC1.1

2.2.3 UC2: CRUD Vorgang

Main Actor	Admin & Planer
Preconditions	Der Admin / Planer wählt ein Vorhaben aus. Der Planer kann nur Vorhaben auswählen, die ihm zugewiesen sind.
Main Success Scenario	Der Admin / Planer kann die Vorgänge für ein Vorhaben bearbeiten, löschen und neue Vorgänge erstellen. Ein Vorgang hat ein Arbeitsvolumen in Stunden definiert. Das Start- / Enddatum muss innerhalb des dazugehörigen Vorhabens definiert sein. Ein Vorgang kann auch Abhängigkeiten zu anderen Vorgängen haben. Ausserdem kann definiert werden, ob eine Aktivität quittiert werden muss oder nicht.

Tabelle 2.3: UC2

2.2.4 UC3: Eigenschaften erstellen

Main Actor	Admin
Main Success Scenario	Der Admin kann Eigenschaften erstellen.

Tabelle 2.4: UC3

2.2.5 UC4: Ressourcen bearbeiten

Main Actor	Admin
Main Success Scenario	Der Admin kann Resourcen ansehen und die Fähigkeiten bearbeiten. Bei Änderungen sollen die Messages neu validiert werden.

Tabelle 2.5: UC4

2.2.6 UC5: Kalenderansicht

Main Actor	User (Ressource)
Main Success Scenario	Der User sieht in einer Kalenderansicht (Monats- / Wochenansicht) seine geplanten Vorhaben. Für jeden Allokation wird ein Eintrag angezeigt.

Tabelle 2.6: UC5

UC5.1: Einsatzplan PDF Export

Main Actor	User (Ressource)
Preconditions	Der User befindet sich in der Kalenderansicht.
Main Success Scenario	Es kann ein Einsatzplan als PDF exportiert werden. Darin werden alle Vorhaben und Vorgänge der aktuell angezeigten Zeitachse aufgelistet, für die er zugewiesen wurde.

Tabelle 2.7: UC5.1

2.2.7 UC6: Quittierung Vorgang

Main Actor	Planer (Admin) & User (Ressource)
Preconditions	Ein Vorhaben wurde ausgewählt.
Main Success Scenario	Der Planer kann auf einem Vorgang die Quittierung aktivieren. Ist die Quittierung aktiv, kann der User seine Allokation zu diesem Vorgang auf erledigt setzen.

Tabelle 2.8: UC6

2.2.8 UC7: Projektplan PDF Export

Main Actor	Planer (Admin)
Preconditions	Ein Vorhaben wurde ausgewählt.
Main Success Scenario	Für ein Vorhaben wird ein Projektplan als PDF generiert. Im Projektplan ist in einer Gantt-Ansicht ersichtlich, wann die Vorgänge von welcher Ressource abgearbeitet werden.

Tabelle 2.9: UC7

2.2.9 UC8: Vorhaben planen

Main Actor	Planer (Admin)
Preconditions	Der Planer hat mindestens ein zugewiesenes Vorhaben.
Main Success Scenario	Der Planer kann ein Vorhaben planen, indem er Ressourcen den Vorgängen zuweist. Für die Zuweisung müssen sowohl ein Start- und Enddatum als auch die Arbeitsstunden erfasst werden.
Success Guarantee	Eine Allokation zwischen Ressource und Vorgang wurde erstellt. Falls es bei der optionalen Validierung Fehler gab, wird eine Meldung mit dem Typ Information, Warnung oder Fehler erstellt.

Tabelle 2.10: UC8

2.2.10 UC9: Ansicht wechseln

Main Actor	Planer (Admin)
Main Success Scenario	Der Planer kann zwischen einer Gesamt-, einer Vorhaben- oder einer Ressourcenansicht wechseln und darauf seine Planung ausführen.

Tabelle 2.11: UC9

2.2.11 UC10: Meldung akzeptieren

Main Actor	Planer (Admin)
Preconditions	Eine optionale Validierung hat einen Fehler entdeckt und eine Meldung wurde für eine Allokation erstellt.
Main Success Scenario	Der Planer kann eine Meldung bewusst in Kauf nehmen und sie akzeptieren.
Success Guarantee	Die Meldung wird in der Planungsansicht nicht mehr angezeigt.

Tabelle 2.12: UC10

2.3 Nicht funktionale Anforderungen

Die Informationen zu den nicht funktionalen Anforderungen wurden aus dem Dokument "Anforderungen 2BIT" [25] von Raphael Ritter entnommen.

2.3.1 Responsiveness

"Die gesamte Applikation soll responsive nach dem Konzept Mobile First ausgestaltet werden. Die effektive Planungsansicht muss mindestens auf einem Tablet vernünftig bedienbar sein." [25]

2.3.2 Browserkompatibilität

”Es müssen keine legacy Browser wie Internetexplorer unterstützt werden. Die Applikation muss aber auf allen gängigen Browsern (Chrome, Firefox, Edge, Safari) einwandfrei funktionieren.” [25]

2.3.3 Antwortzeiten

”Grundsätzlich soll immer innerhalb einer Sekunde eine Antwort vom Server erhalten und angezeigt werden. Beim Aufbau der Planungsansicht darf es bis zu 5 Sekunden dauern, wobei die bereits geladenen sofort angezeigt werden sollen. Zudem soll eine Platzhalteransicht angezeigt werden, die dann abhängig vom Ladefortschritt ersetzt wird.” [25]

2.3.4 Lokalisation

”Die gesamte Applikation muss übersetzbar sein in verschiedene Sprachen. Initial wird mit Deutsch und Englisch (Default) gearbeitet.” [25]

2.3.5 Design

”Es sollen die Guidelines [17] von Material Design eingehalten werden.” [25]

2.4 Optionale Validierung

Die optionale Validierung generiert Messages zu einem Vorhaben, Vorgang, Allokation oder Ressource. Diese Messages dienen dazu, dem Planer Informationen über allfällige Komplikationen oder sonstige, für den Planer interessante Informationen zu liefern. Welche Szenarien die optionale Validierung überprüfen sollen, hat Raphael Ritter im Dokument ”Anforderungen 2BIT” [25] bereits beschrieben:

1. ”Es kann eine Allokation für die Tage erstellt werden, an denen die Ressource nicht verfügbar ist. Anschliessend wird eine entsprechende Fehlermeldung angezeigt.” [25]
2. ”Es können mehr Arbeitsstunden alloziert werden, als die Ressource zur Verfügung stellt. Es wird aber eine entsprechende Fehlermeldung angezeigt.” [25]
3. ”Es können alle Ressourcen gewählt werden. Falls die gefordeten Eigenschaften nicht erfüllt sind, wird ein Fehler angezeigt.” [25]
4. ”Eine Ressource kann parallel gebucht werden. Die parallelen Buchungen werden aber mit einer Fehlermeldung versehen.” [25]
5. ”Ein Vorhaben kann überbucht / unterbucht sein, wodurch eine Warnung auf dem Projekt angezeigt wird.” [25]
6. ”Falls eine verbuchte Ressource aufgrund von Absenzen nicht verfügbar ist, wird eine entsprechende Fehlermeldung angezeigt.” [25]
7. ”Falls eine Ressource nicht voll ausgelastet ist, wird eine entsprechende Warnung angezeigt.” [25]
8. ”Falls eine Allokation an einem Feiertag vorgenommen wird, wird eine entsprechende Warnung angezeigt.” [25]

Nachfolgend wird definiert, durch welche Aktionen ein Szenario ausgeführt wird, welche Entitäten davon betroffen sind und durch welche Änderungen ein Konflikt behoben werden kann.

2.4.1 Szenario 1

Dieses Szenario kann auftreten, wenn eine Allokation für eine Ressource erstellt oder das Start-/Enddatum einer Allokation editiert wurde. Die daraus resultierende Message betrifft die Ressource, die Allokation, den Vorgang und das Vorhaben. Diese Meldung kann durch das Löschen oder Editieren des Start-/Enddatums der Allokation behoben werden. Oder für die Ressource wird der/die fehlende/n WorkDay/s erstellt.

Spezialfälle Wenn ein WorkDay einer Ressource gelöscht wird, kann dieses Szenario auf bereits existierende Allokationen zutreffen. Der Spezialfall kann mit den gleichen Schritten behoben werden wie der Normalfall.

Use Cases Das Szenario wird durch folgende Use Cases ausgelöst.

- UC8: Vorhaben planen

2.4.2 Szenario 2

Die Validation dieses Szenarios wird durch das Erstellen oder Editieren der WorkHours einer Allokation ausgeführt. Die daraus entstandene Message betrifft die Ressource der bearbeiteten Allokation, alle Allokationen der Ressource, alle Vorgänge, die mit den betroffenen Allokationen verknüpft sind und zuletzt noch alle Vorhaben, denen die Vorgänge untergeordnet sind. Die Message kann durch das Löschen oder Reduzieren der WorkHours einer betroffenen Allokation gelöst werden. Zusätzlich kann dies auch durch Erhöhen der WorkHours oder erstellen zusätzlicher WorkDays der Ressource behoben werden.

Spezialfälle Wenn ein WorkDay einer Ressource gelöscht wird oder die WorkHours einer Ressource reduziert werden, kann dieses Szenario auch auftreten. Der Spezialfall kann mit den gleichen Schritten behoben werden wie der Normalfall.

Use Cases Das Szenario wird durch folgende Use Cases ausgelöst.

- UC8: Vorhaben planen

2.4.3 Szenario 3

Die Validation dieses Szenarios wird durch das Erstellen einer Allokation oder durch das Editieren der Fähigkeiten eines Vorganges oder einer Ressource ausgeführt. Die daraus entstandene Message betrifft die Allokation, den Vorgang und das Vorhaben. Die Message kann durch das Löschen der Allokation, das Editieren der Skills eines Vorganges oder durch das Editieren der Skills einer Ressource behoben werden.

Spezialfälle Wenn einer Ressource einen Skill entfernt oder hinzugefügt wird, kann dieses Szenario auch auftreten. Der Spezialfall kann mit den gleichen Schritten behoben werden wie der Normalfall.

Use Cases Das Szenario wird durch folgende Use Cases ausgelöst.

- UC2: CRUD Vorgang
- UC4: Ressourcen bearbeiten
- UC8: Vorhaben planen

2.4.4 Szenario 4

Dieses Szenario kann auftreten, wenn eine Allokation für eine Ressource erstellt oder das Start-/Enddatum einer Allokation editiert wurde. Die daraus entstehende Message betrifft die Ressource, die Allokationen, die einen überschneidenden Zeitraum haben, die Vorgänge dieser Allokationen und das Vorhaben. Diese Message kann durch das Editieren des Start-/Enddatum der Allokation behoben werden.

Use Cases Das Szenario wird durch folgende Use Cases ausgelöst.

- UC8: Vorhaben planen

2.4.5 Szenario 5

Dieses Szenario kann auftreten, wenn ein Vorgang für ein Vorhaben erstellt oder gelöscht wird oder die WorkingHours eines Vorganges oder Vorhabens geändert werden. Die daraus entstehende Message betrifft das Vorhaben. Diese Message kann durch das Erstellen, Löschen oder Editieren der WorkingHours eines Vorganges oder des Vorhabens gelöst werden.

Use Cases Das Szenario wird durch folgende Use Cases ausgelöst.

- UC1: Vorhaben erstellen / löschen
- UC1.1: Vorhaben bearbeiten
- UC2: CRUD Vorgang

2.4.6 Szenario 6

Dieses Szenario kann auftreten, wenn eine Allokation für eine Ressource erstellt oder das Start-/Enddatum dieser Allokation editiert wurde. Die daraus entstehende Message betrifft die Ressource, die Allokationen, der Vorgang und das Vorhaben. Diese Message kann durch das Editieren des Start-/Enddatum der Allokation behoben werden, oder für die Ressource wird die Absenz entfernt.

Spezialfälle: Wenn eine neue Absenz für die Ressource erstellt wird, kann dieses Szenario auf existierende Allokationen zutreffen. Der Spezialfall kann mit den gleichen Schritten behoben werden wie der Normalfall.

Use Cases Das Szenario wird durch folgende Use Cases ausgelöst.

- UC8: Vorhaben planen

2.4.7 Szenario 7

Dieses Szenario kann auftreten, wenn eine Allokation für die Ressource erstellt, gelöscht oder die WorkingHours editiert werden. Die daraus entstehende Message betrifft die Ressource. Diese Message kann durch das Editieren der WorkingHours einer Allokation der Ressource, das Erstellen oder Löschen eines WorkingDays der Ressource oder durch das Editieren der WorkingHours der Ressource behoben werden.

Spezialfälle: Wenn ein WorkingDay der Ressource erstellt oder gelöscht oder die Working-Hours eines WorkingDays editiert wird, kann dieses Szenario auch auftreten. Der Spezialfall kann mit den gleichen Schritten behoben werden wie der Normalfall.

Use Cases Das Szenario wird durch folgende Use Cases ausgelöst.

- UC8: Vorhaben planen

2.4.8 Szenario 8

Dieses Szenario kann auftreten, wenn eine Allokation erstellt oder das Start-/Enddatum einer Allokation geändert wird. Die daraus entstehende Message betrifft die Allokation, den Vorgang und das Vorhaben. Diese Message kann durch das Editieren des Start-/Enddatum der Allokation behoben werden.

Use Cases Das Szenario wird durch folgende Use Cases ausgelöst.

- UC8: Vorhaben planen

Kapitel 3

Analyse

3.1 Scope Definition

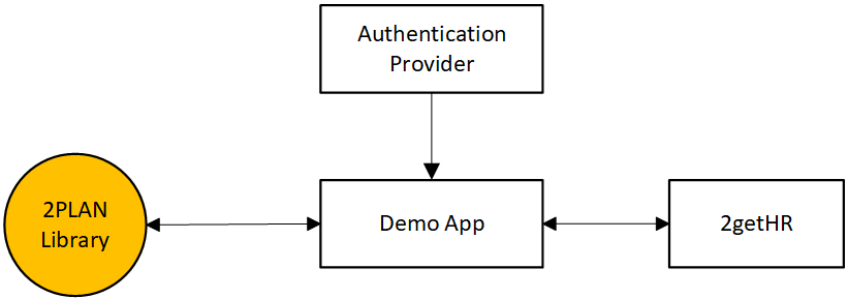


Abbildung 3.1: Context-Diagramm

Die von uns entwickelte 2PLAN Library wird in der Demo-Applikation eingebunden. Über einen Authentication Provider stellt die Demo-Applikation sicher, dass der Benutzer eingeloggt ist. Von der Demo-Applikation erhält dann die Library Informationen über die Benutzerrolle und die ID.
2getHR bildet das Backend ab. Es verwaltet unter anderem die Vorhaben, die Mitarbeiter mit ihrem Arbeitspensum, die Arbeitstage, die Abwesenheiten und Feiertage.
Die Demo-Applikation steuert den ganzen Zugriff auf die 2getHR API und leitet alle benötigten Informationen an die 2PLAN Library weiter.

Nachfolgend wird beschrieben, wie wir die Funktionalitäten und die Views auf die Library und die Demo-Applikation aufteilen. Die Aufteilung wird zusätzlich in den Mockups farblich gekennzeichnet.

3.1.1 Scope Library

Die Library enthält folgende Funktionalitäten:

- die Darstellung und Events für die Kalkulation von Konflikten. Konflikte werden sowohl in Dialogen als auch in einer separaten View dargestellt.
- die Dienstplanansicht einer Ressource. Dazu gehört der Kalender und alle dazugehörigen Funktionalitäten.
- die Dialoge, um eine Allokation zu erstellen und zu bearbeiten.

- die Gantt Ansicht inklusive Filter für die Planung von einem Vorhaben und für die Ressourcenansicht.
- die Funktionen für den PDF-Export.
- ein Binding zur Verfügung stellen, damit Vorgang erstellen/bearbeiten von der Applikation zur Verfügung gestellt werden kann in der Vorhabensansicht des Planers.
- der Kalender für die Gesamtansicht und alle dazugehörigen Funktionalitäten.

3.1.2 Scope Demo-Applikation

Die Demo-Applikation enthält folgende Funktionalitäten:

- die gesamte Navigation, um Views mit unseren Komponenten aus der Library anzuzeigen.
- Views und Funktionalitäten für die Admin Funktionen. Dies entspricht der Stammdatenverwaltung (Vorhaben, Vorgang, Ressource).
- für den Planer wird die Vorhabensansicht mit der Auflistung der Vorhaben, die Detailansicht von einem Vorhaben und die Dialoge, um einen Vorgang zu erstellen oder zu bearbeiten, in der Demo-Applikation implementiert.
- die Logik für die optionale Validierung.

3.2 Domainanalyse

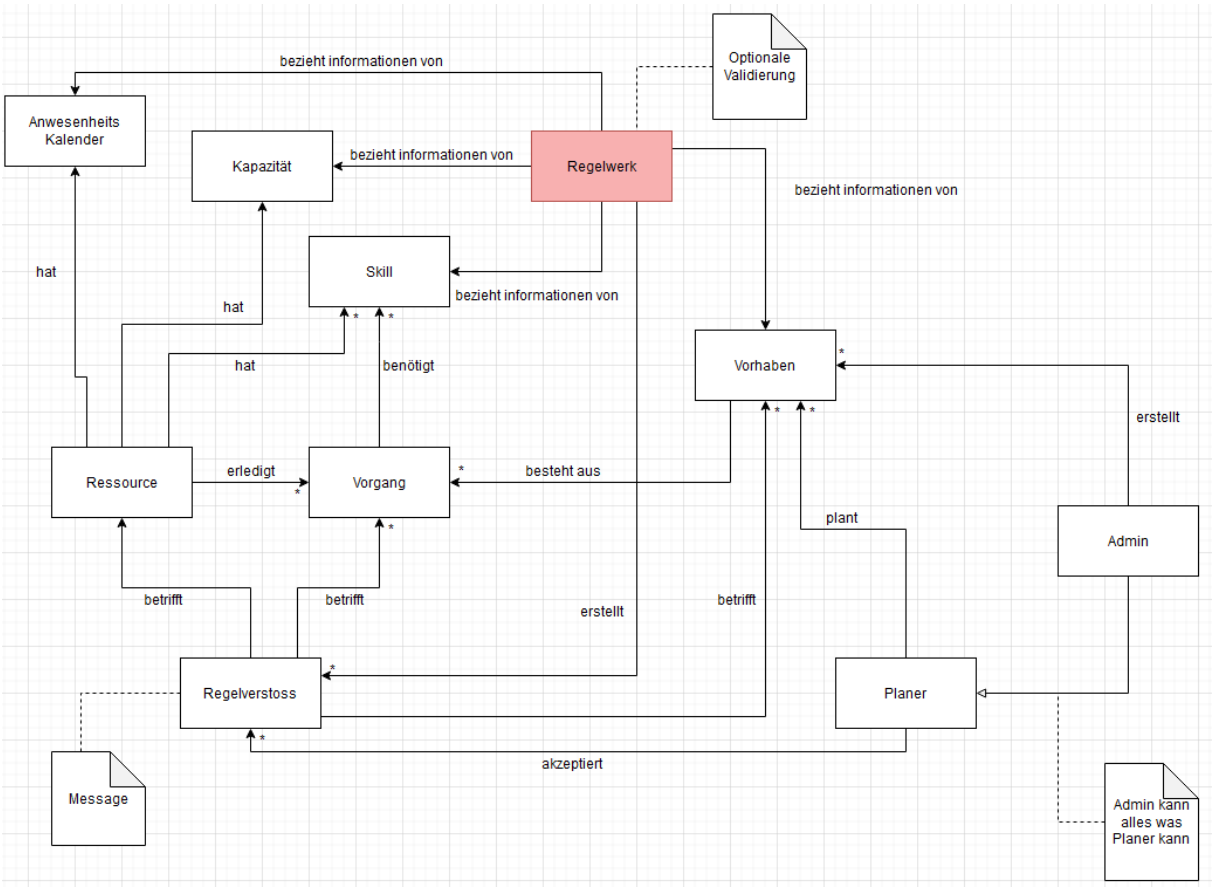


Abbildung 3.2: Domainmodel

Optionale Validierung: Die optionale Validierung wird immer angestossen, wenn Änderungen erfolgen. Zu diesen Änderungen gehört das Editieren der Ressource (Absenz, Anwesenheit, Skills), das Editieren eines Vorganges (Skills, Volumen), das Editieren des Vorhabens (Budget) und die Zuweisung von Ressourcen zu Allokationen. Falls die optionale Validierung eine der definierten Fälle entdeckt, wird eine Message erstellt, die entweder ein Vorhaben, eine Ressource oder einen Vorgang betrifft.

Planer: Der Planer plant die Vorhaben, die ihm zugeordnet sind. Das bedeutet, er erstellt Vorgänge für das Vorhaben und weist diesen Vorgängen Ressourcen zu. Falls es in seiner Planung durch die optionale Validierung zu einer Fehlermeldung kommt, kann der Planer diese entweder akzeptieren oder durch Änderungen in der Planung beheben.

Admin: Der Admin kann alles, was der Planer kann. Zusätzlich ist der Admin dafür verantwortlich die Vorhaben zu erstellen und dem Planer zuzuweisen.

3.3 Zustandsdiagramm

3.3.1 Vorhaben

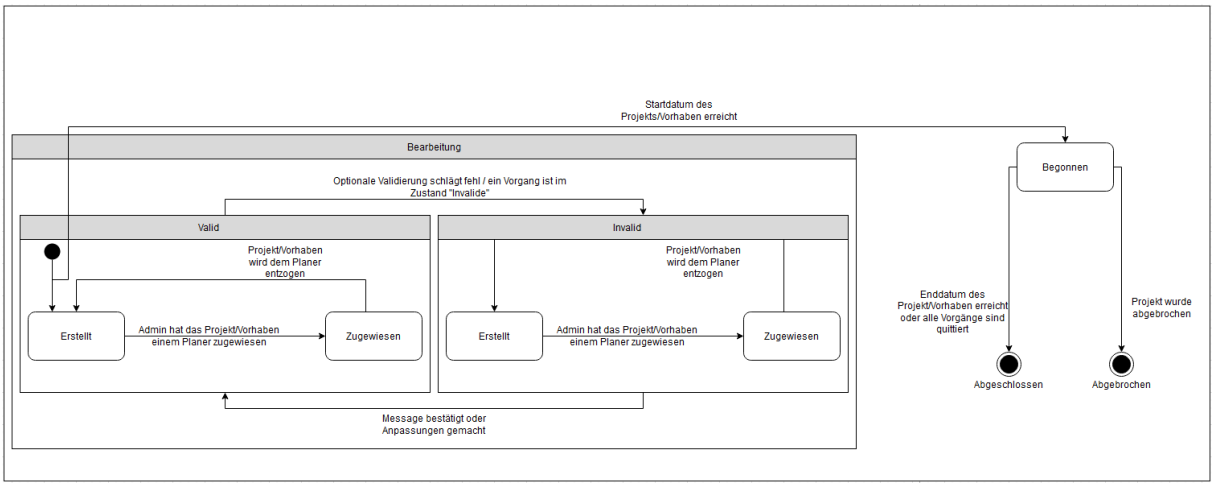


Abbildung 3.3: Zustandsdiagramm Vorhaben

Bearbeitung: Dieser Status ist der Hauptstatus. Direkt nachdem ein ”Admin” ein Vorhaben erstellt hat, befindet sich dieses Vorhaben in diesem Zustand. In diesem Zustand kann das Vorhaben von einem ”Admin” immer bearbeitet werden.

Valid & Invalid: Dies sind zwei Subzustände des ”Bearbeitung”-Zustandes. ”Valid” ist dabei der Startzustand. Wenn die optionale Validierung auf einem Vorgang oder einer Allokation des Vorhabens oder auf dem Vorhaben selbst fehlschlägt, geht dieses Vorhaben in den Zustand ”Invalid” über. Der ”Planer” kann das Vorhaben wieder in den ”Valid” Zustand bringen, indem er entweder die Message akzeptiert, oder den Konflikt löst. Beim Übergang in den jeweils anderen Zustand, bleibt der Subzustand derselbe.

Erstellt: ”Erstellt” ist ein Subzustand der ”Valid” und ”Invalid” Zuständen und ist der Einstiegspunkt nach dem ein Vorhaben erstellt wird. In diesem Zustand ist es dem Administrator möglich, das Vorhaben einem User mit der Rolle ”Planer” zu überweisen. Durch diesen Schritt kommt das Vorhaben in den Zustand ”Zugewiesen”. Der ”Admin” kann dem ”Planer” das Vorhaben auch wieder entziehen, wodurch das Vorhaben wieder in den Status ”Erstellt” übergeht.

Zugewiesen: Wie ”Erstellt” ist ”Zugewiesen” auch ein Subzustand von den ”Valid” und ”Invalid” Zuständen. In diesem Zustand kann der zugewiesene ”Planer” Vorgänge und Allokationen für das Vorhaben erstellen, bearbeiten und löschen.

Begonnen: ”Begonnen” ist ein Parallelzustand zu ”Bearbeitung”. Wenn das Startdatum eines Vorhabens erreicht wurde, befindet sich das Vorhaben im Zustand ”Begonnen” sowie im Zustand ”Bearbeitung”. Wenn anschliessend das Enddatum erreicht wird oder alle Vorgänge eines Vorhabens quittiert sind, wechselt dieses Vorhaben von diesem Zustand auf den Endzustand ”Abgeschlossen”. In den zweiten Endzustand kommt das Vorhaben, wenn entweder der zugewiesene ”Planer” oder ein ”Administrator” das Vorhaben abbricht.

3.3.2 Vorgang

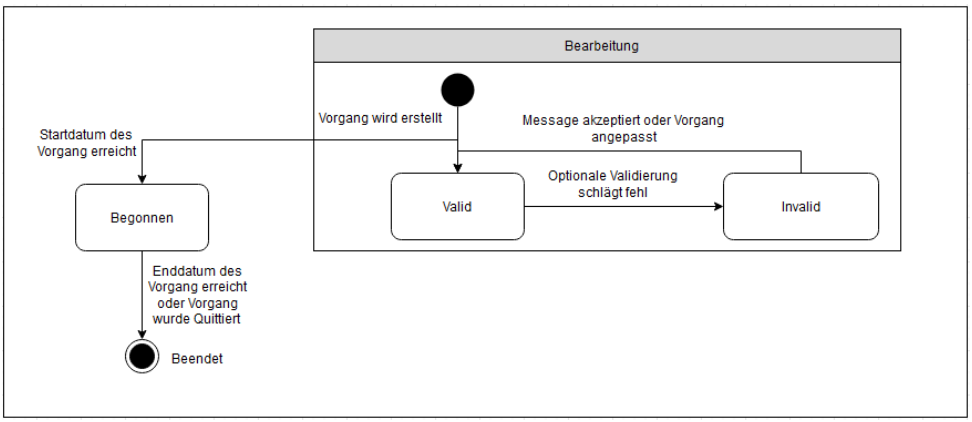


Abbildung 3.4: Zustandsdiagramm Vorgang

Bearbeitung: Dieser Status ist der Hauptstatus. Direkt nachdem ein Vorgang für ein Vorhaben erstellt wurde, befindet sich dieses Vorhaben in diesem Zustand. In diesem Zustand kann der Vorgang bearbeitet werden.

Valid & Invalid: ”Valid” und ”Invalid” sind Subzustände des ”Bearbeitung”-Zustandes. Der Zustand ”Valid” ist der Initialzustand eines Vorganges. Dies kann auf ”Invalid” wechseln, wenn die optionale Validierung für den Vorgang oder den Allokationen dieses Vorganges fehlschlägt. Der Vorgang kann wieder in den Zustand ”Valid” gebracht werde, wenn die Meldung akzeptiert oder gelöst wurde.

Begonnen: ”Begonnen” ist ein Parallelzustand zu ”Bearbeitung”. Wenn das Startdatum eines Vorganges erreicht wurde, befindet sich der Vorgang im Zustand ”Begonnen” sowie im Zustand ”Bearbeitung”. Wenn anschliessend das Enddatum erreicht wird oder alle Allokationen eines Vorganges quittiert sind, wechselt dieser Vorgang von diesem Zustand auf den Endzustand ”Abgeschlossen”.

3.3.3 Ressource

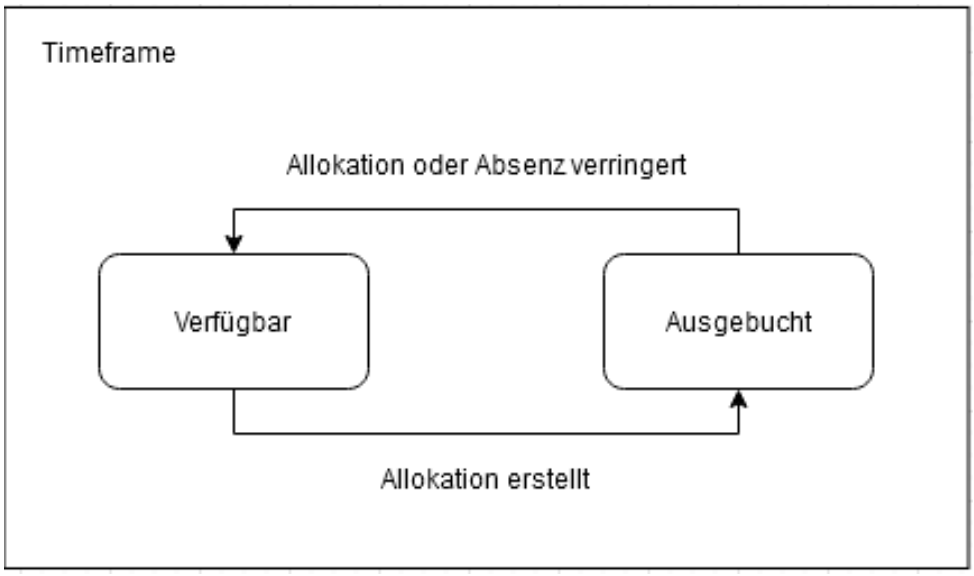


Abbildung 3.5: Zustandsdiagramm Ressource

Die Zustände einer Ressource können nur in einem Zeitfenster angegeben werden. Eine Ressource hat in einem Zeitfenster eine bestimmte Verfügbarkeit. Wird diese Verfügbarkeit von den Allokationen und Absenzen der Ressource komplett aufgebraucht, ist die Ressource für dieses Zeitfenster komplett ausgebucht und wird in der Auswahl eines Vorgangs entsprechend angezeigt. Falls eine Allokation oder Absenz gelöscht oder verringert wird, geht diese Ressource wieder in den Zustand ”Verfügbar”, vorausgesetzt die Ressource ist nun nicht mehr komplett ausgebucht.

Kapitel 4

Evaulierung

Für die Kalenderansicht und das Gantt-Diagramm möchten wir externe Libraries verwenden. In diesem Kapitel wird die Evaluierung der Library dokumentiert.

4.1 Kriterienkatalog

Für die Evaluierung wurden folgende Kriterien verwendet:

- die Library ist Open Source und die Lizenz schränkt die Benutzung nicht ein.
- die Library ist Lightweight.
- die optische Erscheinung ist gut. Es wird Material Design angewendet oder sieht ähnlich aus.
- die Komponente ist responsive.
- die Library ist einfach anzuwenden und eine Dokumentation ist vorhanden.
- Erweiterungen können einfach eingebaut werden.
- unsere benötigten Funktionalitäten sind durch die Library abgedeckt.

4.2 Kalenderansicht

4.2.1 Fullcalendar

Vorteile	**Nachteile**
+ Bietet alle Funktionen, die wir für den Kalender haben möchten. + Lightweight, man kann genau die Plugins einbinden, die man benötigt. + Standard Support ist gratis mit MIT License. + Alle Funktionen, die wir vom Kalender benötigen gehören zum Standard Support. + Sehr gute Beispiele und Dokumentation.	- Erweiterungen sind schwer einzubauen. - Ist nicht responsive.

Tabelle 4.1: Evaluierung Fullcalendar

Fazit: Fullcalendar [13] ist vom optischen und von den Funktionalitäten her sehr ansprechend. Leider ist die Library nicht responsive und Erweiterungen sind nicht einfach einzubauen, was unserer Meinung nach ein sehr grosser Nachteil ist.

4.2.2 angular-calendar

Vorteile	Nachteile
+ Open Source, MIT License. + Lightweight. + Schönes Design. + Ist einigermassen responsive. + Library sollte gut erweiterbar sein.	- Es sind nicht ganz alle benötigten Funktionen vorhanden. - Design ist nicht genau so wie wir es uns vorgestellt haben.

Tabelle 4.2: Evaluierung angular-calendar

Fazit: Angular-calendar [3] setzt optisch die Funktionalitäten nicht genau so um wie wir uns das vorgestellt haben. Dennoch ist das Design sehr schön und auch einigermassen responsive. Ausserdem steht der Sourcecode zur Verfügung, ist gut strukturiert und ist dementsprechend erweiterbar.

4.2.3 Auswertung

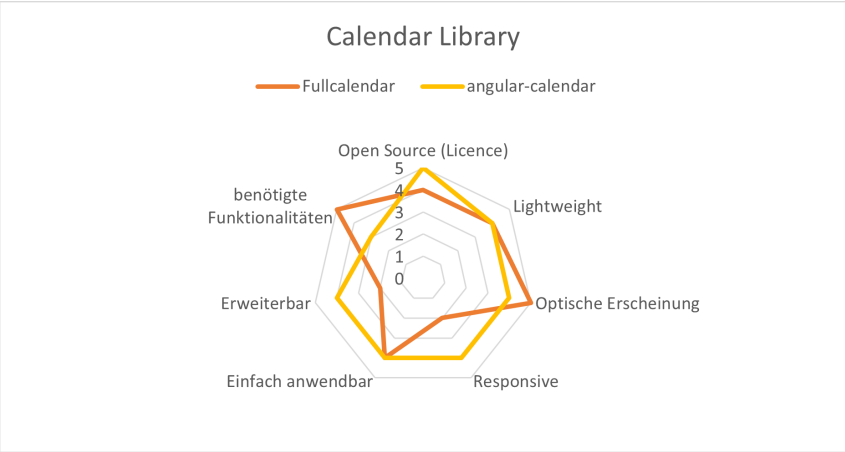


Abbildung 4.1: Calendar-Library Evaluation

Nachdem wir die Vor- und Nachteile miteinander verglichen haben, sind wir zum Entschluss gekommen, die angular-calendar [3] Library zu verwenden. Wir legen sehr viel Wert darauf, dass Erweiterungen eingebaut werden können, was uns bei dieser Library einfacher erscheint.

4.2.4 Entscheidung nach Prototyp

Beim Implementieren des Prototyps stellte sich heraus, dass die angular-calendar [3] Library doch nicht geeignet ist. Da die Library ihre Kalkulationen basierend auf Datentypen einer

weiteren Library erstellt, müssten Änderungen tief im Core der Library implementiert werden. Dies bedeutet, dass man nicht einfach in der View Änderungen implementieren kann, sondern auch gleich die Datenstrukturen anpassen müsste. Diese Anpassungen würden einen enormen Aufwand bedeuten und der Erfolg ist nicht garantiert. Aus diesem Grund haben wir doch noch die Fullcalendar [13] Library in Betracht gezogen. Wie bereits vermutet, zeigte sich dort ein ähnliches Problem mit dem Anpassen. Nachdem wir noch kurz nach einer alternativen Lösung in Ionic2-Calendar [19] und Telerik Calendar [28] gesucht haben, fassten wir den Beschluss, dass die Kalenderansicht komplett von uns implementiert wird. So können wir auch die Anzeige auf unsere Daten zuschneiden und müssen nicht mit generalisierten Datentypen arbeiten.

4.3 Gantt Diagramm

4.3.1 ngx-time-scheduler

Vorteile	Nachteile
+ Open Source, MIT License.	- Nicht responsive.
+ Einfach anwendbar, gute Dokumentation.	- Kein Material Design.
+ CSS pro Model möglich.	- Locale funktioniert nicht zu 100%.
+ Relativ aktuell, verwendet Angular 9.	- Bietet nicht alle Funktionen, die wir benötigen. Zum Beispiel gibt es keine Filter.
+ Kann gut erweitert werden.	
+ Lightweight.	

Tabelle 4.3: Evaluierung ngx-time-scheduler

Fazit: Der ngx-time-scheduler [26] ist eine einfache Angular Timeline Scheduler Library und liefert eine gute Basis. Die Komponente müsste noch überarbeitet und erweitert werden. Der Code sieht jedoch verständlich und gut erweiterbar aus.

4.3.2 angular-gantt

Vorteile	Nachteile
+ Hat viele Funktionen, die wir benötigen.	- Es funktionieren nicht alle Funktionen.
+ Schönes Design.	- Sehr viel Code. Bräuchte viel Einarbeitungszeit vor allem bei Anpassungen.
+ Eine kurze Dokumentation ist vorhanden.	- Verwendet AngularJs und müsste zuerst auf Angular 11 konvertiert werden.
+ Open Source, MIT License.	

Tabelle 4.4: Evaluierung angular-gantt

Fazit: Die angular-gantt [14] Library bietet viele Funktionen an, die wir brauchen, jedoch funktionieren nicht alle. Ausserdem ist der Code mit AngularJs geschrieben und müsste von uns konvertiert werden, was bestimmt ein grosser Aufwand ist.

4.3.3 gantt-schedule-timeline-calendar

Vorteile	Nachteile
+ Hat viele Funktionen, die wir benötigen. + Sieht sehr gut aus. + Ist einigermassen responsive.	- Kann nur gratis verwendet werden in Non-Commercial Projects die der MIT / ISC Lizenz unterliegen. Der gesamte Sourcecode (Backend und Frontend) muss im Internet verfügbar sein. - Die Software darf nicht erweitert werden.

Tabelle 4.5: Evaluierung gantt-schedule-timeline-calendar

Fazit: Aufgrund der Lizenzbedingungen bietet gantt-schedule-timeline-calendar[15] keine Gratisversion für uns an. Ausserdem gibt es in der Lizenzbeschreibung weitere strenge Regeln. Ansonsten wäre die Library sehr gut.

4.3.4 Auswertung

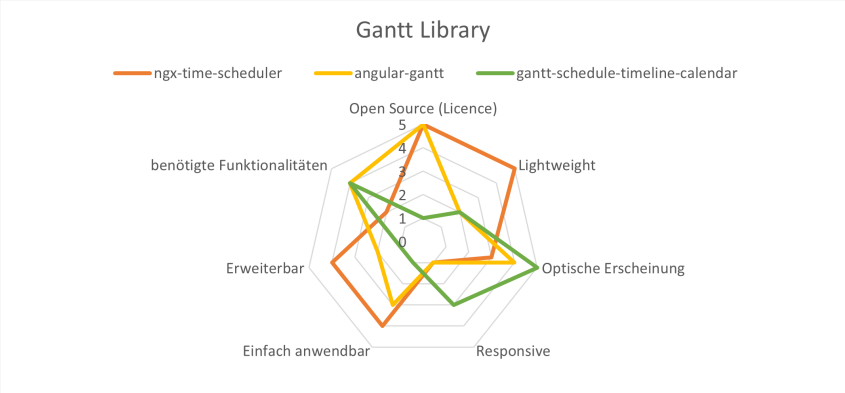


Abbildung 4.2: Gantt-Library Evaluation

Nachdem wir die Vor- und Nachteile miteinander verglichen haben, sind wir zum Entschluss gekommen, die ngx-time-scheduler [26] Library zu verwenden. Die Library liefert uns eine gute Basis und kann relativ einfach erweitert werden, was uns sehr wichtig war. Ausserdem ist sie Open Source, Lightweight und verwendet eine einigermassen aktuelle Angular Version. Deshalb sollte ein Upgrade keine grosse Sache werden.

Kapitel 5

Architektur & Design Spezifikation

5.1 Datenmodell

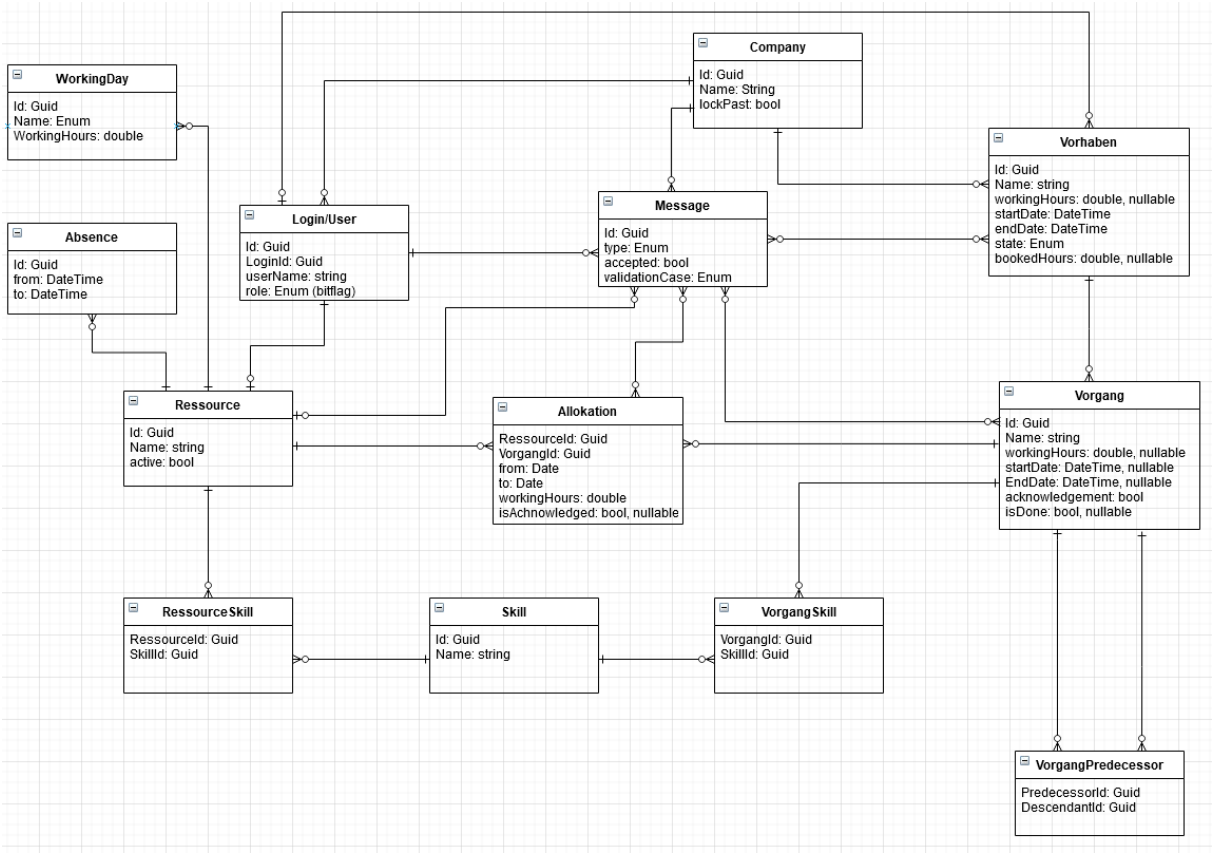


Abbildung 5.1: Datenmodell

Der Anwesenheitskalender und die Kapazität aus dem Domainmodel wird aus den WorkingDays und Absences für die Ressource berechnet. Die Message kann ausserdem auch eine Allokation betreffen und nicht nur Vorhaben, Vorgänge und Ressourcen. Die ValidationGroup bezieht sich auf das Szenario der optionalen Validierung, welches dazu geführt hat, dass die betroffene Meldung erstellt wurde. Dadurch kann bei einer Anpassung entschieden werden, welche Messages behoben werden konnten.

5.2 Layer Diagramm

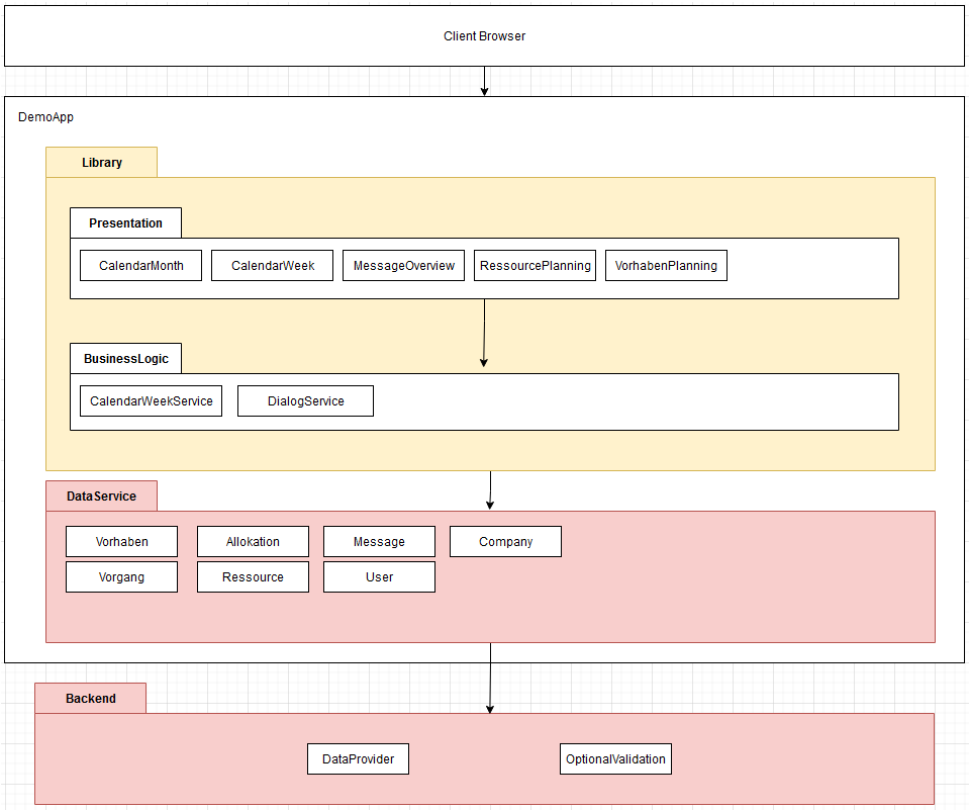


Abbildung 5.2: Architektur Layer

Der Client Browser ruft die Demo-Applikation auf, die unsere Library einbindet. Die Komponenten, die unsere Library zur Verfügung stellt, werden von der Applikation angezeigt. Einige Komponenten der Library, zum Beispiel die "CalendarWeek" Komponente, haben ihren eigenen Business Service, welche die übergebenen Daten auswerten, damit diese dargestellt werden können. Die Library erhält die Daten über die Data Services der Demo-Applikation. Der Data Service Layer bezieht die Daten wiederum von einer API des .NET Core Backends. Neben dem Verwalten der Daten führt das Backend die optionale Validierung durch.

5.3 Backend

Der Zugriff auf die Daten der Datenbank läuft über eine .NET Core API. Die API-Definition wird von uns festgelegt und die Implementierung der API wird von 2BIT vorgenommen.

5.3.1 API-Architektur

Die Architektur der API gibt vor, wie das Data Management der Angular-Applikation gehandhabt wird. Aus diesem Grund mussten wir uns überlegen, wie die API gestaltet werden soll und welche Einflüsse dies auf unser Data Management hat. Genauer ging es dabei um zwei Architekturarten.

Graph-Architektur

In der Graph-Architektur werden alle Entitäten der Datenbank separat geladen. Anschliessend werden die Daten auf dem Client verbunden. Bei dieser Variante kann man viele Requests auf den Server einsparen, da die Daten nur am Anfang geladen werden und danach nur noch Modifizierungen verschickt werden. Ein grosser Nachteil dieser Architektur ist jedoch die Komplexität und Synchronisation der Daten. Ausserdem müssten wir uns bei diesem Ansatz in ein neues Framework einarbeiten wie zum Beispiel NgRx [11], welches uns das Datenmanagement abnehmen würde. Durch die Einarbeitung würde für uns ein grosser Mehraufwand entstehen.

Flow-Architektur

In der Flow-Architektur teilt man die Applikation nach Flows auf und erstellt dann für jeden Flow ein API Endpoint. Wenn ein User dann einen Flow ausführt, werden alle Daten, die für diesen Flow gebraucht werden, geladen. Da die Daten auf die Flows abgestimmt werden, können die Objekte, die von der API verschickt werden, von der Struktur auf der Datenbank abweichen. Der Nachteil dieser Architektur ist, dass mehr Requests an den Server geschickt werden und die Performance schlechter ist. Ausserdem müssen die Flows definiert und das API basierend auf den Flows aufgesetzt werden. Wenn die Flows zu einem späteren Zeitpunkt ändern, müsste somit sehr wahrscheinlich auch eine Änderung am API vorgenommen werden.

Entscheidung

Anfangs haben wir uns für die Graph-Architektur entschieden. Jedoch hatten unser Betreuer und unser Kunde eine andere Ansicht bezüglich der Implementation des Data Management. Nach einem Austausch gingen wir nochmals über unsere Entscheidung und machten uns die Konsequenzen davon bewusst. Auch wenn wir unsere erste Lösung für eleganter halten, war die zusätzliche Komplexität für uns ein grosses Kontra. Da wir beide kaum Erfahrung mit Angular haben, war die Programmiersprache bereits eine grosse Herausforderung für uns. Durch den Einsatz eines Data-Management Frameworks wie NgRx [11] hätten wir ein weiteres, für uns unbekanntes Framework angewendet. Da dies für uns zu riskant war, haben wir uns schlussendlich für die Flow-Architektur entschieden.

Optionale Validierung

Ursprünglich sollte die optionale Validierung auf dem Client statt finden. Mit unserem Architekturentscheid der API ist dies jedoch nicht mehr möglich. Der Grund dafür ist, dass wir auf dem Client nicht alle Daten geladen haben, um die optionale Validierung durchzuführen. Aus diesem Grund wird die optionale Validierung nun auch auf dem Server in .NET Core implementiert. Jedoch wird dies von uns übernommen und nicht von 2BIT, auch wenn dies auf dem Server ist.

5.3.2 API-Definition

Zu Beginn hatten wir die API mit der Graph-Architektur definiert. Das API-Design mit der Graph-Architektur wird noch im Anhang E aufgeführt. Aus vorheriger Begründung haben wir schlussendlich die definitive Architektur gemeinsam mit einem Entwickler von 2BIT, namens Maxim, nach Flow-Architektur definiert.

Maxim implementierte dann das API und stellte eine Swagger Ansicht zur Verfügung. Ausserdem gab er uns ein Konfigurationsfile für Nswag. So können wir bei API-Anpassungen

über Swagger schnell kontrollieren, ob die Änderungen korrekt sind und mit nswag die API-Anbindung in Angular einfach und fehlerfrei generieren lassen.

Die API-Pfade sind alle nach dem Schema `/api/{Entity}` aufgebaut. `Entity` ist dabei die Entität, auf die zugegriffen wird, also Vorgang, Allokation, Vorhaben, etc. (in Englisch). Für jeden dieser Pfade gibt es einen Endpunkt, korrespondierend zu den HTTP Request Typen GET, DELETE, PUT und POST. Ausserdem gibt es für jede Entität noch weitere Funktionen, um etwas von einem spezifischen Objekt dieser Entität zu laden. In diesem Fall wird der Pfad mit der ID dieses Objektes und der gewünschten Funktion ergänzt beispielsweise `/api/Activities/{id}/allocations`. Dieser Beispielendpunkt liefert alle Allokationen eines bestimmten Vorganges zurück.

Kapitel 6

Technologien

Nachfolgend werden alle Technologien, die für 2PLAN verwendet wurden, aufgelistet und beschrieben.

6.1 Angular

Angular [2] ist ein clientseitiges Web-Framework, das auf TypeScript [31] basiert. Die Web-Applikationen werden aus Komponenten zusammengesetzt. Eine Komponente besteht aus einer TypeScript-Klasse mit einem `@Component` Dekorator, einem HTML-Template und einem optionalen Set von CSS Styles.

Die Angular Component Library 2PLAN wurde mit Angular 11 entwickelt. Dafür wurden verschiedene Angular Konzepte wie Pipes, Direktiven oder Content projection angewendet.

6.2 TypeScript

TypeScript [31] ist eine auf JavaScript basierende Open Source Language, die zusätzlich statische Typendefinitionen ermöglicht. Jeder gültige JavaScript Code ist auch ein gültiger TypeScript Code. TypeScript Code wird über einen TypeScript Compiler in JavaScript Code umgewandelt und läuft überall, wo auch JavaScript läuft.

6.3 Protractor

Protractor [24] ist ein E2E Test Framework für Angular. Protractor Tests interagieren mit der Applikation genau so, wie es ein normaler Anwender tun würde, da die Tests in einem echten Browser ausgeführt werden.

Für die Hauptfeatures von 2PLAN haben wir E2E Tests implementiert um sicherzustellen, dass die Features funktionieren.

6.4 .NET Core

Das Backend der Demo-Applikation wurde mit .NET Core [1] entwickelt. .NET Core ist ein Framework, das zur Implementierung von Anwenungdsprogrammen verwendet werden kann.

6.5 NPM

NPM [21] ist ein Paketmanager, der es ermöglicht, Packages zu teilen oder private Packages zu verwalten.

Die Library 2PLAN wurde als privates Package zu NPM hinzugefügt. Dadurch können Endanwender unsere Library mittels `npm install @2bitgmbh/planning-component` installieren und anwenden.

6.6 Azure DevOps

Mithilfe von Azure DevOps [9] lassen sich Softwareprojekte verwalten. Azure DevOps bietet einige Features an, um die Zusammenarbeit in einem Team zu vereinfachen. Beispielsweise kann mit Azure Repos die Quellcodeverwaltung übernommen werden, mit Azure Pipelines kann eine Continuous Integration definiert werden und über Azure Boards kann das Projekt geplant und die Arbeit nachverfolgt werden.

6.7 Gitlab

Gitlab [16] dient als Versionsverwaltungssystem für Softwareprojekte auf der Basis von Git.

Für unsere Bachelorarbeit haben wir Gitlab nur verwendet, um Sprints zu planen und die Arbeitszeiten zu tracken.

Kapitel 7

Implementierung

7.1 Internationalization / Localization

Eine unserer nicht funktionalen Anforderungen besagt, dass die gesamte Applikation in verschiedene Sprachen übersetzbar sein muss (siehe Kapitel 2.3.4).

Für die Umsetzung haben wir einerseits die Third Party Library ngx-translate [29] und i18n [5] von Angular in Betracht gezogen. Da 2BIT selbst auch ngx-translate verwendet und die Library bereits in der Demo-Applikation eingerichtet war, haben wir uns entschieden, ebenfalls ngx-translate zu verwenden.

Mit ngx-translate werden die Übersetzungen zur Laufzeit geladen. Für jede implementierte Sprache gibt es eine JSON Datei. Die Übersetzung kann auf drei verschiedene Varianten gelesen werden:

TranslateService

```
translate.get('HELLO', {value: 'world'}).subscribe((res: string) => {
    console.log(res); //=> 'hello world'
});
```

TranslatePipe

```
<div>{{ 'HELLO' | translate }}</div>
```

TranslateDirective

```
<div [translate]="'HELLO'" [translateParams]="{value: 'world'}"></div>
```

Da die Übersetzung in der Library auf JSON Dateien basiert, müssen diese auch im NPM Package mitgeliefert und schlussendlich in der Applikation eingebunden werden. Um die Dateien dem NPM Package hinzuzufügen, wurde im ng-package.json der Library folgendes ergänzt:

```
{
    "assets": [ "./assets" ],
}
```

So wird der Assets Ordner, in dem sich die Dateien befinden, in das NPM Package mit eingebunden. Welche Schritte zusätzlich in der Applikation unternommen werden müssen, um die Übersetzung der Library anwenden zu können, wird in der Anwenderdokumentation (siehe Anhang D) beschrieben.

7.2 Angular Material Komponenten

Bei der Implementierung von 2PLAN und der Demo-Applikation haben wir darauf geachtet, ein modernes und anspruchsvolles Design zu verwenden. Dafür haben wir die UI-Komponenten von Angular Material [6] verwendet.

Um die Komponenten nicht überall zu importieren, haben wir ein `MaterialModule` erstellt. In diesem Modul haben wir alle häufig genutzten Material Komponenten importiert und exportiert. Durch den Export sind die Komponenten überall zugreifbar wo `MaterialModule` importiert wird. Das `MaterialModule` haben wir anschliessend im `SharedModule` importiert und exportiert.

7.3 Optionale Validierung

Wir haben für die optionale Validierung zwei der acht Szenarien im .NET Core Backend implementiert. Dabei ging es vor allem darum, die Struktur der optionalen Validierung festzulegen und Messages zu erstellen, da die Messages nicht im UI erstellt werden können. Bei der Implementation achteten wir sehr auf die Erweiterbarkeit, da die restlichen Szenarien von 2BIT implementiert werden. Es wurde eine Klasse implementiert, die das Regelwerk abbildet und Referenzen auf die Klassen der Validierungsszenarien besitzt. In diesem "Regelwerk" gibt es Methoden, welche die Szenarien ausführen je nachdem, was gemacht wurde. Wird zum Beispiel eine neue Allokation erstellt, kann eine korrelierende Funktion des "Regelwerks" ausgeführt werden und alle betroffenen Szenarien werden ausgeführt. Das "Regelwerk" wird über Dependency Injection den Controllern zur Verfügung gestellt.

7.3.1 Regelwerk

```
public class OptionalValidationService : IOptionalValidationService
{
    private readonly ResourceNotAvailableCase _resourceNotAvailableCase;

    public OptionalValidationService(AppDbContext context)
    {
        _resourceNotAvailableCase = new ResourceNotAvailableCase(context);
    }

    public async Task CheckAllocationCases(Guid userId, Allocation allocation, bool isUpdate =
        false)
    {
        await _resourceNotAvailableCase.CheckSzenario(userId, allocation, isUpdate);
    }

    public async Task DeleteAllocationMessages(Allocation allocation)
    {
        await _resourceNotAvailableCase.DeleteOfCaseForAllocation(allocation);
    }
}
```

7.3.2 Szenario

Die Klasse `ResourceNotAvailableCase` bildet das Szenario 1 und 6 der optionalen Validierung ab (siehe Kapitel 2.4).

```
public class ResourceNotAvailableCase
{
    private readonly AppDbContext _context;
    private readonly MessageCreator _messageCreator;
```

```
public ResourceNotAvailableCase(AppDbContext context)
{
    _context = context;
    _messageCreator = new MessageCreator(_context);
}

public async Task CheckSzenario(Guid userId, Allocation allocation, bool isUpdate = false)
{
    var resource = _context.Resources.Find(allocation.ResourceId);
    if (resource == null)
        return;

    if (isUpdate)
    {
        await CheckAndDeleteExistingMessages(allocation);
    }
    await CheckAbsences(userId, resource, allocation);
    await CheckWorkDays(userId, resource, allocation);
}

public async Task DeleteOfCaseForAllocation(Allocation allocation)
{
    _context.Messages.RemoveRange(_context.Messages.Where(m => m.ValidationCase ==
        EF.Models.Enums.ValidationCases.ResourceNotAvailable && m.Allocations.Any(a => a.Id
        == allocation.Id)));
    await _context.SaveChangesAsync();
}

private async Task CheckAndDeleteExistingMessages(Allocation allocation)
{
    _context.Messages.RemoveRange(_context.Messages.Where(m => m.ValidationCase ==
        EF.Models.Enums.ValidationCases.ResourceNotAvailable && !m.Accepted &&
        m.Allocations.Any(a => a.Id == allocation.Id)));
    await _context.SaveChangesAsync();
}

private async Task CheckAbsences(Guid userId, Resource resource, Allocation allocation)
{
    var overlappingAbsences = resource.Absences?.Where(a => CheckAbsenceInAllocation(a,
        allocation));
    foreach(var absence in overlappingAbsences)
    {
        await _messageCreator.CreateNewMessage(userId, allocation,
            EF.Models.Enums.MessageTypes.Error,
            EF.Models.Enums.ValidationCases.ResourceNotAvailable, absence.From >
            allocation.From ? absence.From : allocation.From, absence.To > allocation.To ?
            allocation.To : absence.To);
    }
}

private async Task CheckWorkDays(Guid userId, Resource resource, Allocation allocation)
{
    var date = allocation.From.Date;
    while (date.Date <= allocation.To.Date)
    {
        if (!resource.WorkingDays.Any(wd => wd.Name == date.DayOfWeek))
        {
            await _messageCreator.CreateNewMessage(userId, allocation,
                EF.Models.Enums.MessageTypes.Error,
                EF.Models.Enums.ValidationCases.ResourceNotAvailable, date);
        }
        date = date.AddDays(1);
    }
```

```
    }

    private bool CheckAbsenceInAllocation(Absence absence, Allocation allocation)
    {
        if ((absence.From < allocation.To && absence.To >= allocation.To) ||
            (absence.From <= allocation.From && absence.To > allocation.From))
            return true;
        return false;
    }
}
```

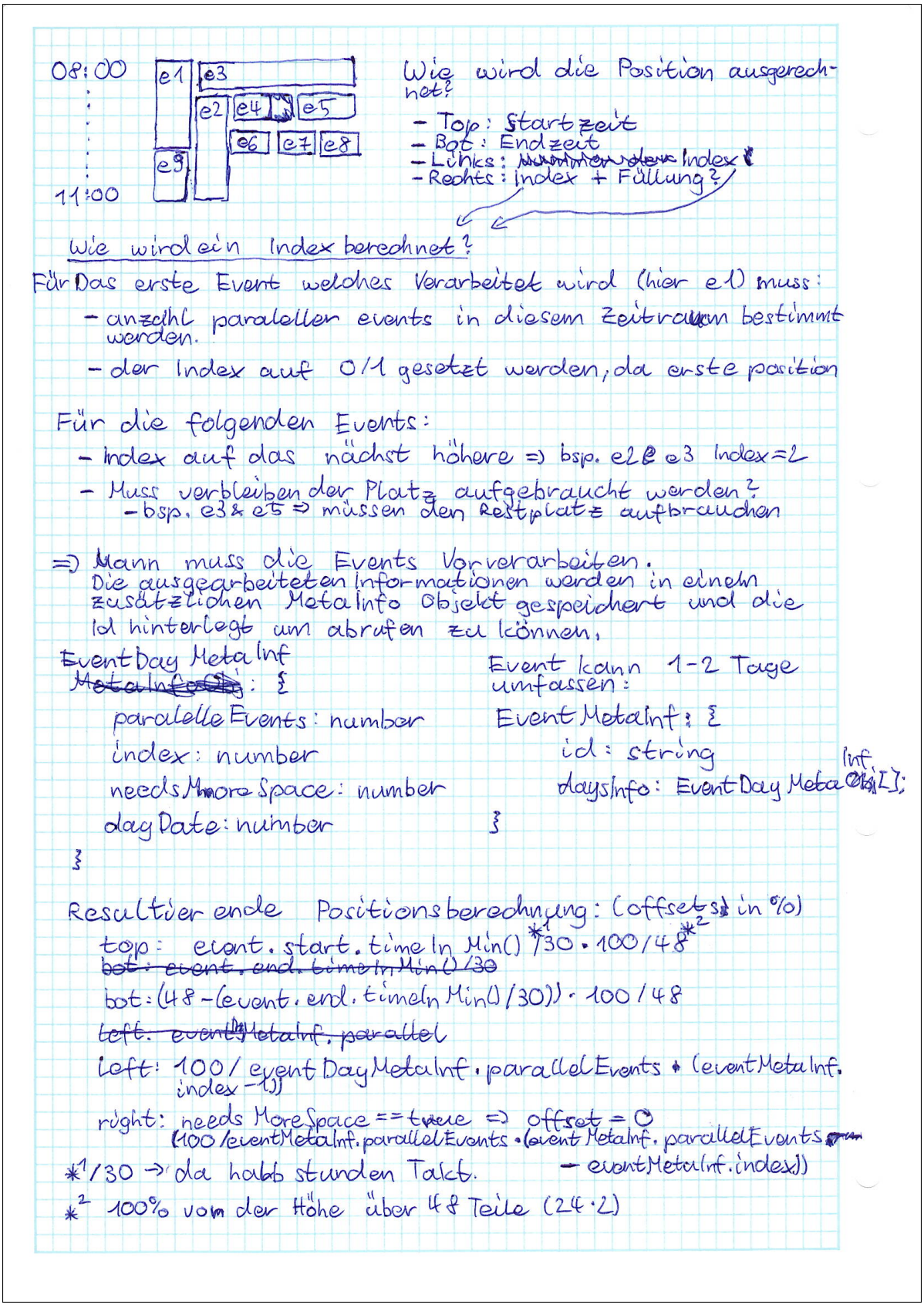
7.3.3 Erweiterung

Beim Implementieren eines neuen Szenarios erstellt man zuerst eine Klasse, welche die Logik für dieses Szenario implementiert. Danach fügt man die Referenz auf die Szenario-Klasse dem ”Regelwerk” hinzu und führt das Szenario in den entsprechenden Methoden aus. Falls für das Szenario noch keine korrespondierende Methode existiert, muss diese im ”Regelwerk” hinzugefügt und im Interface angepasst werden. Nun kann man im verantwortlichen Controller die Methode des ”Regelwerks” ausführen und das neue Szenario wird ausgeführt.

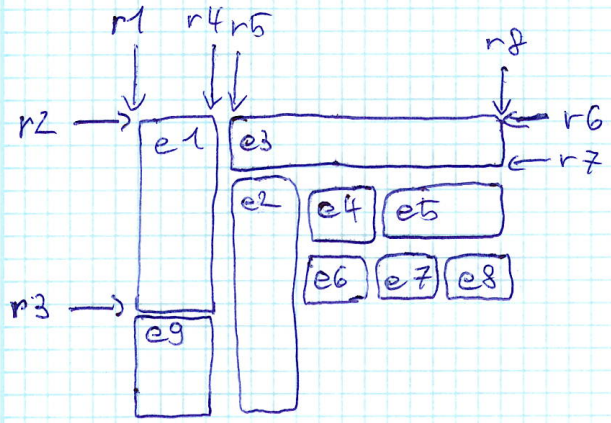
7.4 Kalender-Algorithmus

In der Wochenansicht des Kalenders ist es nicht so einfach, die Events (Allokationen) anzuzeigen. Das Problem liegt darin, dass sich die Events in Zeiträumen überschneiden können. Diese Komplexität lässt sich nicht mit Styling lösen wie in der Monatsansicht. Die Idee ist nun, die Positionierung für die Events zu berechnen. Für diese Berechnung benötigen wir einen Algorithmus, der die Positionierung in Abhängigkeit zu den jeweils anderen Events der gleichen Zeitspanne ausrechnet. Wie dieser Algorithmus entwickelt wurde, wird anhand unserer Notizen gezeigt.

7.4.1 Notizen


08:00
11:00
e1
e3
e2
e4
e5
e6
e7
e8
e9
Wie wird die Position ausgerechnet?
- Top: Startzeit
- Bot: Endzeit
- Links: ~~Nummer~~ Index
- Rechts: Index + Füllung?
Wie wird ein Index berechnet?
Für Das erste Event welches Verarbeitet wird (hier e1) muss:
- anzahl paralleler events in diesem Zeitraum bestimmt werden.
- der Index auf 0/1 gesetzt werden, da erste position
Für die folgenden Events:
- Index auf das nächst höhere => bsp. e2 & e3 Index=2
- Muss verbleiben der Platz aufgebraucht werden?
-bsp. e3 & e5 => müssen den Restplatz aufbrauchen
=> Mann muss die Events Vorverarbeiten.
Die ausgearbeiteten Informationen werden in einem zusätzlichen MetaInfo Objekt gespeichert und die Id hinterlegt um abrufen zu können.
EventDay MetaInf
~~MetaInfoEvent~~ : {
parallele Events: number
index: number
needsMoreSpace: number
dayDate: number
}
Event kann 1-2 Tage umfassen:
EventMetaInf: {
id: string
daysInfo: EventDayMetaInf[];
}
Resultier ende Positionsberechnung: (offsets in %)
top: event.start.timeIn Min() *1/30 * 100/48 *2
~~bot: event.end.timeIn Min()/30~~
bot: (48 - (event.end.timeIn Min()/30)) * 100/48
~~left: eventDayMetaInf.parallel~~
left: 100/ eventDayMetaInf.parallelEvents * (eventMetaInf.index - 1))
right: needsMoreSpace == true => offset = 0
(100/eventMetaInf.parallelEvents * (eventMetaInf.parallelEvents - eventMetaInf.index))
*1/30 -> da halbb stunden Takt.
*2 100% von der Höhe über 48 Teile (24*2)

Skizze:



r1: Index = 1, parallel Events = 5
100/5 · (1−1) = 0

r2: time/30 · 100/48

r3: (48 − (time/30)) · 100/48

r4: index = 1, parallel Events = 5, needs More Space = false
100/5 · (5−1) ⇔ [illegible] 80 (100−80 = 20%
→ von e1 übernommen, da grösser "width")

r5: Index = 2, parallel Events = 5,
100/5 · (2−1) = 20% (ist gleich r4 (offset right von e1)).

r6: time/30 · 100/48

r7: (48 − (time/30)) · 100/48

r8: Index = 2, parallel Events = 5, needs More Space = true
needs More Space == true ⇒ 0

7.4.2 Code

MetaInf Klassen

```
export class EventMetaInf {
  public id: string | undefined;
  public days: EventDayMetaInf[];

  constructor() {
    this.days = [];
  }
}

export class EventDayMetaInf {
  public dayDate: number | undefined;
  public parallelEvents: number | undefined;
  public index: number | undefined;
  public needsMoreSpace: boolean;
  constructor() {
    this.needsMoreSpace = false;
  }
}
```

Algorithmus

In dieser Funktion werden die Metadaten der Events berechnet, die für die Positionierung benötigt werden.

```
public initWeekSchedule(events: IAllocationDto[], currentDay: Date, eventMetaInfs:
    EventMetaInf[]) {
    for (const event of events) {
      const metaInf = new EventMetaInf();
      const metaObj = new EventDayMetaInf();
      metaObj.parallelEvents = this.getMaxConcurrentEvent(event, events, currentDay.getDate());
      metaObj.dayDate = currentDay.getDate();
      metaInf.id = event.id;
      metaInf.days.push(metaObj);
      if (eventMetaInfs.find(e => e.id === event.id)) {
        eventMetaInfs.find(e => e.id === event.id).days.push(metaObj);
      } else {
        eventMetaInfs.push(metaInf);
      }
    }
    for (const event of events) {
      const metaObj = eventMetaInfs.find(e => e.id === event.id).days.find(d => d.dayDate ===
          currentDay.getDate());
      const sameSpanEvents = this.getSameSpanEvents(events, event, currentDay);
      if (sameSpanEvents.length === 1) {
        metaObj.index = 1;
        metaObj.parallelEvents = 1;
        continue;
      }
      let maxParalellEvents = metaObj.parallelEvents;
      const usedIndexes: number[] = [];
      sameSpanEvents.forEach(e => {
        const temp = eventMetaInfs.find(emi => emi.id === e.id).days.find(d => d.dayDate ===
            currentDay.getDate());
        if (temp.parallelEvents > maxParalellEvents) {
          maxParalellEvents = temp.parallelEvents;
        }
        if (temp.index) {
          usedIndexes.push(temp.index);
        }
```

```
      });
      metaObj.parallelEvents = maxParalellEvents;
      for (let i = 1; i <= metaObj.parallelEvents; i++) {
        if (!usedIndexes.find(idx => idx === i)) {
          metaObj.index = i;
          break;
        }
      }
    }

    for (const event of events) {
      const metaInf = eventMetaInfs.find(e => e.id === event.id).days.find(d => d.dayDate ===
          currentDay.getDate());
      const sameSpanEvents = this.getSameSpanEvents(events, event, currentDay);
      if (sameSpanEvents.length >= 1) {
        let needsMoreSpace = true;
        sameSpanEvents.forEach(e => {
          const temp = eventMetaInfs.find(emi => emi.id === e.id).days.find(d => d.dayDate ===
              currentDay.getDate());
          if (temp.index > metaInf.index) {
            needsMoreSpace = false;
          }
        });
        if (needsMoreSpace && metaInf.index < metaInf.parallelEvents) {
          metaInf.needsMoreSpace = true;
        }
      }
    }

    return eventMetaInfs;
}
```

MetaInf Auswertung

Diese Funktion wird aus dem HTML aufgerufen und bindet den Return-Wert an den Style des Event-Elements. Die Funktion nimmt die berechneten Metadaten des Events und gibt die Position dieses Events zurück.

```
public calcInset(event: IAllocationDto, dayDate: number) {
    const eventMetaInf = this.eventMetaInfs.find(e => e.id === event.id).days.find(d =>
        d.dayDate === dayDate);
    const startMin = this.calendarWeekService.getMinOfDay(event.from, dayDate, true);
    const endMin = this.calendarWeekService.getMinOfDay(event.to, dayDate, false);
    const offsetTop = this.calcTop(Math.floor(startMin / 30));
    const offsetBot = this.calcBot(Math.ceil(endMin / 30));

    let offsetRight: string;
    if (!eventMetaInf.needsMoreSpace) {
      offsetRight = (100 / eventMetaInf.parallelEvents * (eventMetaInf.parallelEvents -
          eventMetaInf.index)) + "%";
    } else {
      offsetRight = "0%";
    }

    const offsetLeft = (100 / eventMetaInf.parallelEvents * (eventMetaInf.index -
        1)).toString() + "%";
    return `top: ${offsetTop}; right: ${offsetRight}; bottom: ${offsetBot}; left:
        ${offsetLeft};`;
}
```

7.4.3 Auswertung

Der entwickelte Algorithmus funktioniert und stellt die überschneidenden Events nebeneinander an. Die Schwachstelle des Algorithmus liegt jedoch in der Performance. Wenn man sich den Code der `initWeekSchedule` Methode ansieht, wird man gleich auf die drei Schlaufen aufmerksam. Diese drei Schlaufen werden für die Vorbereitung, die Verarbeitung und die Nachbearbeitung der Events benötigt. Um den Algorithmus zu optimieren, müssten diese drei Schritte fusioniert werden. Dies ist jedoch nicht so einfach, wie es zuerst den Eindruck macht. Zum Beispiel nehmen wir einen Event, der von 8 Uhr - 10 Uhr stattfindet. Der Event liegt im Zeitraum eines anderen Events, der von 7 Uhr - 9 Uhr dauert. Für das Event von 7 Uhr - 9 Uhr haben wir nur das Event von 8 Uhr - 10 Uhr im gleichen Zeitraum und können somit die Breite durch zwei teilen. Wenn es allerdings mehrere Events im Zeitraum von 9 Uhr - 10 Uhr gibt, finden diese Events parallel zu unserem ersten Event statt und haben somit auch Auswirkungen auf die Breite des Events von 7 Uhr - 9 Uhr. Obwohl die Performance eine wichtige Rolle spielt, ist es sehr unwahrscheinlich, dass eine Ressource so viele Events in einer Woche hat, dass dies grosse Auswirkungen auf die Performance haben würde. Aus diesem Grund haben wir den Algorithmus nicht mehr verbessert und uns auf andere Features fokussiert.

Verbesserungen

Um den Algorithmus zu verbessern, könnte das vorgängig erwähnte Problem wahrscheinlich mit einer Rekursion behoben werden. Der Ablauf würde noch nach dem gleichen Prinzip stattfinden. Jedoch würden die drei Schritte nicht mehr für alle Events nacheinander durchgegangen werden, sondern pro Zeitraum eines Events. Zum Beispiel würden diese Schritte gleich für das erste Event und alle Events in diesem Zeitraum durchgeführt werden. Wenn wir davon ausgehen, dass nicht alle Events am gleichen Zeitpunkt stattfinden, könnte dies die Performance verbessern, obwohl der Overhead für den Speicherbedarf steigt. Weiterhin könnte der Algorithmus asynchron und auch für die anliegenden Wochen bereits ausgeführt werden. So müsste der User nicht bei jedem Wochenwechsel warten.

7.5 Gantt-Algorithmus

Nachfolgend wird der Algorithmus dokumentiert, der in der Gantt-Ansicht benötigt wird, um die Positionierung der Allokationen zu berechnen. Der grösste Teil des Algorithmus stammt aus der ngx-time-scheduler [26] Library. Wir haben uns aber dennoch intensiv damit auseinandergesetzt und kleine Änderungen eingebaut.

7.5.1 Models

Als Section wird das bezeichnet, was im Gantt-Diagramm in der ersten Spalte abgefüllt ist, also zum Beispiel ein Vorgang oder eine Ressource. Ein Item widerspiegelt eine Allokation.

SectionItem Für jede Section wird ein `SectionItem` erstellt. Darin ist die eigentlich Section enthalten und für jede dazugehörige Allokation wird ein `ItemMeta` erstellt. Das Property `minRowHeight` enthält die Gesamthöhe einer Section, die anhand der untereinander angezeigten Items berechnet wird. Sie entspricht mindestens 40px.

```
export class SectionItem {
    public section: Section;
    public minRowHeight: number;
    public itemMetas: ItemMeta[];

    constructor() {
```

```
        this.itemMetas = new Array<ItemMeta>();
    }

    public numberOfLines() {
        const lineHeight = 21;
        const number = this.minRowHeight / lineHeight;
        return Math.floor(number);
    }
}
```

ItemMeta Für jede Allokation, die im definierten Zeitbereich sichtbar ist, wird ein `ItemMeta` erstellt und dem `SectionItem` zugewiesen.

```
export class ItemMeta {
  public item: Item;
  public isStart: boolean;
  public isEnd: boolean;
  public cssTop: number;
  public cssLeft: number;
  public cssWidth: number;

  constructor() {
    this.cssTop = 0;
    this.cssLeft = 0;
    this.cssWidth = 0;
  }
}
```

7.5.2 Berechnung

Zu Beginn wird für jede Section ein `SectionItem` erstellt. Für das `SectionItem` wird die `section` und die `minRowHeight` gesetzt.

Anschliessend wird für alle sichtbaren Items ein `ItemMeta` erstellt und der Section und zu `allVisibleItemMetas` hinzugefügt.

```
public initItemsPerSectionItem(): void {
  const allVisibleItemMetas = new Array<ItemMeta>();

  this.sectionItems.forEach(sectionItem => {
    sectionItem.itemMetas = new Array<ItemMeta>();
    sectionItem.minRowHeight = this.minRowHeight;

    if (this.items) {
      this.items.forEach(item => {
        if (item.sectionID === sectionItem.section.id) {
          let itemMeta = new ItemMeta();
          itemMeta.item = item;
          if (itemMeta.item.start <= this.periodEnd && itemMeta.item.end >= this.periodStart) {
            itemMeta = this.itemMetaCal(itemMeta);
            sectionItem.itemMetas.push(itemMeta);
            allVisibleItemMetas.push(itemMeta);
          }
        }
      });
    }
  });

  const sortedItemsBySection = this.sortItemMetasBySectionId(allVisibleItemMetas);
  this.calCssTop(sortedItemsBySection);
}
```

Mit der Funktion `itemMetaCal` wird berechnet, wie breit das Item prozentual zur Gesamtbreite sein soll. Ausserdem wird der linke Abstand vom Item bis zum Periodenanfang prozentual zur Gesamtbreite berechnet.

```
public itemMetaCal(itemMeta: ItemMeta): ItemMeta {
  const foundStart = moment.max(itemMeta.item.start, this.periodStart);
  const foundEnd = moment.min(itemMeta.item.end, this.periodEnd);

  let widthMinuteDiff = Math.abs(foundStart.diff(foundEnd, "minutes"));
  let leftMinuteDiff = foundStart.diff(this.periodStart, "minutes");
  if (this.showBusinessDayOnly) {
    widthMinuteDiff -= (this.getNumberOfWeekendDays(moment(foundStart), moment(foundEnd)) *
        this.currentPeriod.timeFramePeriod);
    leftMinuteDiff -= (this.getNumberOfWeekendDays(moment(this.periodStart),
        moment(foundStart)) * this.currentPeriod.timeFramePeriod);
  }

  itemMeta.cssLeft = (leftMinuteDiff / this.currentPeriodMinuteDiff) * 100;
  itemMeta.cssWidth = (widthMinuteDiff / this.currentPeriodMinuteDiff) * 100;

  if (itemMeta.item.start >= this.periodStart) {
    itemMeta.isStart = true;
  }
  if (itemMeta.item.end <= this.periodEnd) {
    itemMeta.isEnd = true;
  }

  return itemMeta;
}
```

Zu guter Letzt wird pro Section für jedes Item der CSS Top Wert berechnet. Der Wert ist normalerweise immer 0, ausser wenn es innerhalb einer Section Items gibt, die eine zeitliche Überschneidung haben und deshalb untereinander angezeigt werden sollen. Falls Items untereinander angezeigt werden, wird die `minRowHeight` der Section erhöht.

```
  public calCssTop(sortedItems): void {
    for (const sectionIndex of Object.keys(sortedItems)) {
      for (let i = 0; i < sortedItems[sectionIndex].length; i++) {
        let elemBottom;
        const elem = sortedItems[sectionIndex][i];

        const sortedPrevElem = sortedItems[sectionIndex]
                                            .slice(0, i)
                                            .sort((a, b) => (a.cssTop > b.cssTop) ? 1 : -1);
        for (let prev = 0; prev < i; prev++) {
          const prevElem = sortedPrevElem[prev];
          const prevElemBottom = prevElem.cssTop + this.minRowHeight;
          elemBottom = elem.cssTop + this.minRowHeight;

          if (this.dateRangeOverlaps(prevElem.item.start, prevElem.item.end,
                                     elem.item.start, elem.item.end) && (
            (prevElem.cssTop <= elem.cssTop && elem.cssTop <= prevElemBottom) ||
            (prevElem.cssTop <= elemBottom && elemBottom <= prevElemBottom)
          )) {
            elem.cssTop = prevElemBottom + 1;
          }
        }

        elemBottom = elem.cssTop + this.minRowHeight + 1;
        if (this.sectionItems[Number(sectionIndex)] && elemBottom >
            this.sectionItems[Number(sectionIndex)].minRowHeight) {
          this.sectionItems[Number(sectionIndex)].minRowHeight = elemBottom;
        }
```

```
    }
  }
}
```

Kapitel 8

Testing

8.1 Integration Tests

Da die grösste Priorität bei einer funktionierender Library im Zusammenhang mit den Umsystemen liegt, setzen wir nur Integration Tests und keine Unit Tests (nach Martin Fowler [32]) ein. Ausserdem hatte auch unser Auftraggeber den Wunsch geäussert, dass wir uns vor allem auf Integration Tests mit Angular E2E Tests fokussieren. Da wir Features wie die optionale Validierung in das Backend ausgelagert haben, gibt es zudem fast keine Funktionalitäten, die von Unit Tests profitieren könnten.

Die Integration Tests werden mit dem Angular E2E Testframework Protractor [24] durchgeführt. Bei der Implementierung der Tests haben wir uns auf die Sunny Cases der wichtigsten Use Cases fokussiert. Dazu gehören die Use Cases UC5, UC8, UC9 und UC10 (siehe Kapitel 2.2). Die Sunny Cases drehen sich dabei um die Anzeige der Elementen in der Übersicht, die Anzeige der Details in einem Popup und die Standarddurchführung der Use Cases ohne Konfliktbehandlung. So stellen wir sicher, dass die Integration und die Hauptfeatures der Library funktionieren. Für detailliertere Tests müssten Unit Tests implementiert werden. Die Integration Tests werden bei jedem Commit auf den Dev oder Master durchgeführt. So garantieren wir, dass keine kaputte Version gepublished wird.

8.1.1 Durchführung

Für die Projekt- und Ressourcenplanung wird das Erfassen der Testdaten über das UI gemacht. So kann der komplette Prozess einer Planung geprüft werden. Bei den restlichen Tests werden die Daten durch den Kontext direkt über das Backend API erstellt. Im nachfolgenden Codeabschnitt wird über das API eine Ressource und ein Vorhaben inklusive einem Vorgang erstellt. Anschliessend wird für den Vorgang und die Ressource eine Allokation definiert:

```
beforeAll(async () => {
    context = await SpecContext.new();
    const currentDate = new Date();
    const endDate = new Date(currentDate);
    endDate.setDate(currentDate.getDate() + 4);
    await context.loadUserResource();
    await context.newProject(new TestProject("E2E Test Project", currentDate.toISOString(),
        endDate.toISOString(), 42, 0, 0, context.user.id, "blue"));
    await context.newActivity(new TestActivity("E2E Test Activity", currentDate.toISOString(),
      endDate.toISOString(), 30, context.user.project.id, false, false));
    await context.newAllocation(new TestAllocation(context.user.activity.name,
        context.user.resource.name, currentDate.toISOString(), endDate.toISOString(), 30,
        context.user.resource.id, context.user.activity.id, false));
});
```

Nachdem die Daten aufbereitet wurden, wird die Funktionalität anhand der Angezeigten DOM Elementen getestet. Zum Beispiel wird in einer Listenübersicht geprüft, ob ein Element dieser Liste angezeigt wird:

```
it("Check if message exists", async () => {
    const { asUser } = context;
    await asUser(async () => {
      await browser.get(`${browser.baseUrl}${page.route}`);
      const isPresent = await page.isMessageContainerPresent();
      expect(isPresent).toEqual(true);
    });
});
```

8.2 Cognitive Walkthrough

Da eine gute User Experience ein Hauptziel von 2PLAN ist, war ursprünglich die Idee, mit verschiedenen Endkunden von 2BIT Usability Tests durchzuführen. Leider gerieten wir durch API-Änderungen in Verzug mit der Arbeit. Aus diesem Grund hatten wir zu wenig Zeit, um aussagekräftige Usability Tests durchzuführen, Verbesserungen einzubauen und einen erneuten Usability Tests zu machen, um zu verifizieren, ob unsere Verbesserungen den gewünschten Effekt erbrachten.

Nach Absprache mit unserem Betreuer und 2BIT haben wir uns dazu entschlossen, anstelle der Usability Tests einen Cognitive Walkthrough durchzuführen. Beim Cognitive Walkthrough wird eine Reihe von Tasks abgearbeitet, um das Verständnis der Erlernbarkeit des Systems zu testen. Diesen Cognitive Walkthrough haben wir mit Priska Steiger, einer User Experience Designerin, durchgeführt. Der Fokus lag darauf herauszufinden, ob der Aufbau und der Ablauf der Applikation verständlich ist und welche Design-Optimierungen noch eingebaut werden könnten.

Das Ergebnis des Cognitive Walkthrough ist im Anhang C zu finden.

8.3 Performance Test

8.3.1 Mengengerüst

Die Datenmenge haben wir so gewählt, um die Planung eines Unternehmen für einige Jahre abzudecken. Dabei ist nachfolgendes Mengengerüst entstanden:

- 150 Vorhaben
- pro Vorhaben 10 Vorgänge
- pro Vorgang 10 Allokationen (alle Allokationen haben das gleiche Start- und Enddatum)
- 30 Ressourcen
- 100 Messages

8.3.2 Vorbereitung

Um den Performance Test durchführen zu können, hat uns 2BIT ein Skript mit den entsprechenden Datenmengen aufbereiten. Da das Skript 250 anstatt 150 Vorhaben enthalten hat, haben wir den Performance Test mit mehr Daten durchgeführt als geplant. Insgesamt wurden 25'000 Allokationen generiert was ungefähr 833 Allokationen pro Ressource entspricht.

Um möglichst keine Performanceeinbussen durch Server Requests zu haben, haben wir eine lokale Datenbank erstellt und diese mit den Daten aus dem Skript befüllt. Anschliessend haben wir den Connection String für die Datenbank auf unsere lokale Instanz gesetzt.

8.3.3 Ergebnisse

Bei der Analyse haben wir uns auf die wichtigsten Komponenten fokussiert. Für jede Komponente wurde die Zeit mindestens fünf mal gemessen und anschliessend wurde der Durchschnitt berechnet. In der Tabelle ist jeweils die Differenz zwischen den verschiedenen Abschnitten zu finden. Zusätzlich wird die Gesamtdauer bis zum Anzeigen der Daten aufgelistet. Die Ergebnisse haben einen Toleranzbereich von 0.3 Sekunden.

Die Zeiten, die einen Einfluss auf das Rendering haben, wurden rot markiert. Falls ein Algorithmus die Messung stark beeinflusst, wurde die Zeit orange markiert.

Nachfolgend sind die Spezifikationen des Gerätes zu finden, mit dem die Tests durchgeführt wurden.

Gerät	HP EliteBook x360 1030 G2
Prozessor	Intel Core i7-7600U
Betriebssystem	Windows 10 Pro
Browser	Google Chrome (Version 91.0.4472.77)

Tabelle 8.1: Gerät Performance Test

Vorhabensübersicht

Die Vorhabensübersicht wurde mit 250 Vorhaben getestet.

phgr-project-overview-page	ngOnInit	0ms
	ngAfterViewInit	103ms
phgr-project-list	ngOnInit	122ms
	ngAfterViewInit	681ms
Total		**906ms**

Tabelle 8.2: Performance Vorhabensübersicht

Projektplanung

Die Projektplanung wurde zuerst mit 10 Vorhaben ohne Allokationen getestet.

lib-gantt-activity	ngOnInit	0ms
	Subscribe Data (Ressource and Allocation)	233ms
lib-gantt	ngOnInit	48ms
	start Calculation	1ms
	end Calculation	5ms
	ngAfterViewInit	71ms
Total		**358ms**

Tabelle 8.3: Performance Projektplanung

Anschliessend wurde der Test mit 10 Vorhaben und insgesamt 100 gleichzeitig stattfindenden Allokationen getestet.

lib-gantt-activity	ngOnInit	0ms
	Subscribe Data (Ressource and Allocation)	1s 383ms
lib-gantt	ngOnInit	28ms
	start Calculation	1ms
	end Calculation	7ms
	ngAfterViewInit	140ms
Total		**1s 559ms**

Tabelle 8.4: Performance Projektplanung mit Allokationen

Ressourcenplanung

Die erste Messung für die Ressourcenplanung wurde mit 30 Ressourcen und ohne Allokationen durchgeführt.

lib-gantt-resource	ngOnInit	0ms
	Subscribe Data (Ressource and Allocation)	1s 551ms
lib-gantt	ngOnInit	37ms
	start Calculation	1ms
	end Calculation	65ms
	ngAfterViewInit	361ms
Total		**2s 15ms**

Tabelle 8.5: Performance Ressourcenplanung

Die zweite Messung wurde mit 30 Ressourcen und 25'000 gleichzeitig stattfindenden Allokationen durchgeführt.

lib-gantt-resource	ngOnInit	0ms
	Subscribe Data (Ressource and Allocation)	1s 788ms
lib-gantt	ngOnInit	33ms
	start Calculation	1ms
	end Calculation	6s 332ms
	ngAfterViewInit	7s 901ms
Total		**16s 55ms**

Tabelle 8.6: Performance Ressourcenplanung mit Allokationen

Dienstplan - Monatsansicht

Die Monatsansicht des Dienstplanes wurde mit einer Ressource getestet, die 906 zugewiesene Allokationen hat.

lib-calendar-month	ngOnInit	0ms
	initMonth start	1ms
	initMonth end	8ms
	ngAfterViewInit	364ms
Total		**373ms**

Tabelle 8.7: Performance Dienstplan Monatsansicht

Dienstplan - Wochenansicht

Die Wochenansicht des Dienstplanes wurde zuerst mit einer Ressource getestet, die 906 zugewiesene Allokationen hat. Alle Allokationen gehen über mehrere Tage.

lib-calendar-week	ngOnInit	0ms
	initMonth start	12ms
	initMonth end	14ms
	ngAfterViewInit	706ms
Total		**732ms**

Tabelle 8.8: Performance Dienstplan Wochenansicht

Anschliessend wurde die Wochenansicht noch mit 906 Allokationen getestet, die weniger als einen Tag dauern.

lib-calendar-week	ngOnInit	0ms
	initMonth start	9ms
	initMonth end	14s 173ms
	ngAfterViewInit	107ms
Total		**14s 289ms**

Tabelle 8.9: Performance Dienstplan Wochenansicht mit kurzen Allokationen

8.3.4 Auswertung

Rendering

Wir haben bei diversen Komponenten das Problem, dass das Rendern zu lange dauert. Dies erkennt man, wenn man die Zeiten für `ngAfterViewInit` analysiert. Um dieses Problem zu beheben, könnte Infinite Scrolling oder Virtual Scrolling [18] implementiert werden.

Infinite Scrolling Beim Infinite Scrolling wird eine initiale Datenmenge geladen und angezeigt. Wenn man scrollt und das Ende erreicht hat, werden weitere Daten zur initialen Datenmenge hinzugefügt und angezeigt. Bei dieser Strategie werden also immer wieder Elemente zum DOM hinzugefügt. Dadurch, dass das DOM ständig erweitert wird, steigt der Memory-Bedarf.

Virtual Scrolling Beim Virtual Scrolling werden nur die Elemente, die dem User angezeigt werden, gerendert. Alle andere Elemente, die nicht sichtbar sind, werden aus dem DOM entfernt. Diese Strategie ist vor allem empfehlenswert, wenn die Anwendung öfters auf Mobilgeräten verwendet wird. Angular Material stellt bereits eine Virtual Scroller Komponente [8] zur Verfügung.

Algorithmen

Die Berechnung für die Positionierung der Allokationen bei der Planung und beim Dienstplan benötigen mehrere Sekunden, wenn man mit vielen Daten testet. Hier müssten die Alogrithmen genauer analysiert werden, um zu prüfen, ob und wo Optimierungen eingebaut werden können. Für den Algorithmus im Dienstplan haben wir bereits eine Auswertung und einen dazugehörigen Verbesserungsvorschlag dokumentiert (siehe Kapitel 7.4.3).

8.3.5 Weitere Findings

Nachfolgend werden weitere Findings dokumentiert, die beim Testen mit vielen Daten entdeckt wurden.

Binding

Wenn man den Dienstplan mit vielen Daten testet, funktionieren die Bindings auf die Element Properties innerhalb des `*ngFor` nicht. Für eine Allokation wird dann nur ein grauer Balken angezeigt, ohne die Bezeichnung. Erst wenn man mit der View interagiert, wird `ngAfterViewInit` ausgeführt und die korrekten Daten werden angezeigt.

Progress Spinner

Immer wenn Daten von der Datenbank geladen werden, wird ein Progress Spinner angezeigt. Wenn die Daten bereits auf dem Client vorhanden sind und nur noch verarbeitet und an-

gezeigt werden müssen, wird aktuell kein Progress Spinner angezeigt. Dadurch erhält der Benutzer kein Feedback ob im Hintergrund etwas geladen wird oder nicht. Dieses Verhalten ist gut erkennbar, wenn man die Ressourcenplannung öffnet und anschliessend als Startdatum den 21.01.2021 auswählt. Der Browser reagiert anschliessend nicht mehr, da im Hintergrund der Algorithmus für die Positionierung der Allokationen ausgeführt wird. Mit dem Angular Progress Spinner [7] könnte dieses Problem behoben werden.

Kapitel 9

Code Metriken

9.1 Lines of Code

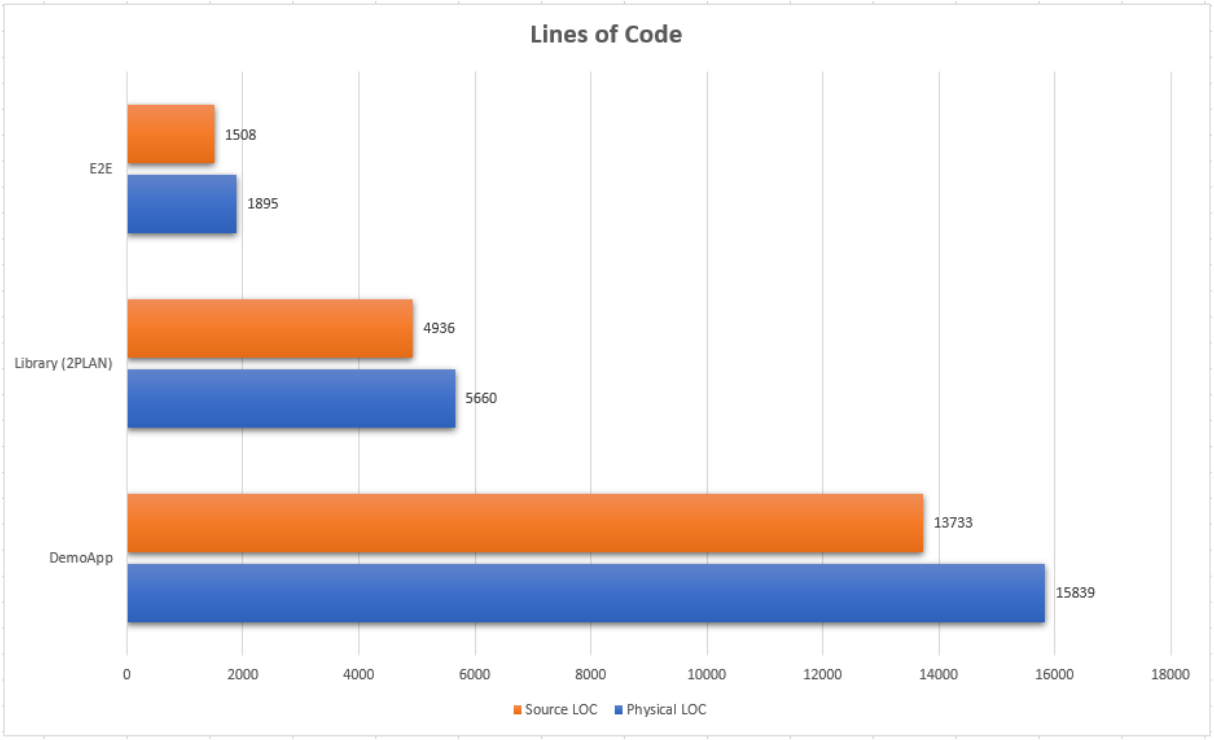


Abbildung 9.1: Lines of Code

Im Endprodukt unserer Bachelorarbeit wurden insgesamt 23'394 Physische Code-Zeilen und 20'177 Logische Code-Zeilen geschrieben. Diese Auswertung wurde mit Sloc [27] erstellt. Ein Entwickler mit 1-5 Jahren Programmiererfahrung schreibt ungefähr 100 Zeilen Code pro Tag [10]. In unserer Arbeit dauerte die Construction Phase 9 Wochen. Da wir pro Woche ca. 20-25 Stunden Aufwand für die Bachelorarbeit hatten, rechnen wir mit 3 Tagen pro Woche. Dies ergibt 27 Tage, an denen wir Code schreiben. Laut der verwendeten Statistik hätten wir somit 2'700 Code Zeilen pro Person geschrieben. Dies entspricht jedoch nicht unserer Auswertung mit Sloc. Natürlich berücksichtigte Sloc die Abgrenzung von generiertem Code und dem Code, der uns 2BIT vorbereitet hatte, nicht. Weiterhin wird noch viel Overhead Code von Angular erwartet. Dazu gehören zum Beispiel Module, Komponenten Deklarationen, Pipe Deklarationen, etc. Durch diese Abgrenzungen nehmen wir an, dass wir nur 1/3 des Codes

für Funktionalitäten implementiert haben. Dies bringt uns auf 3’899 Code-Zeilen pro Person.

9.1.1 Auswertung

Schlussendlich haben wir ca. 1’000 Code-Zeilen mehr pro Person mehr geschrieben. Dies lässt sich auf unsere Unerfahrenheit mit Angular zurückführen. Dadurch haben wir sicher für einige Funktionalitäten mehr Code-Zeilen geschrieben als nötig gewesen wären. Dies war jedoch auch zu erwarten und ist nicht all zu schlimm.

Kapitel 10

Projektmanagement

10.1 Projektorganisation

Severin Amacher wird zusammen mit Silvan Gehrig die Rolle des Scrum Masters einnehmen. Julia Tanner ist verantwortlich für die Kontrolle der Dokumentation. Wir verfolgen das Ziel, dass alle Teammitglieder mit jedem Feature in Berührung kommen. Julia fokussiert sich aber hauptsächlich auf die Planung und Severin auf den Dienstplan.

10.2 Projekt Meetings

Alle zwei Wochen findet am Montag um 17:30 ein Statusmeeting mit dem Betreuer statt. Versetzt dazu gibt es alle zwei Wochen am Montag um 08:30 ein Meeting mit dem Auftraggeber. An diesen Meetings wird der aktuelle Stand anhand der Demo-Applikation präsentiert. Ausserdem werden Probleme, Fragen und das weitere Vorgehen besprochen.

10.2.1 Zwischenpräsentation

Nach dem Meilenstein "End of Elaboration" werden wir eine Zwischenpräsentation halten. Diese dient dazu, um unsere Entscheidungen und die Massnahmen, die wir durchgeführt haben, um die Risiken zu minimieren, zu präsentieren.

10.2.2 Protokollierung

Für Sitzungen, die mit dem Betreuer oder dem Auftraggeber durchgeführt werden, werden wir jeweils ein Sitzungsprotokoll schreiben, sofern wichtige Beschlüsse gefasst wurden.

10.2.3 Teammeeting

Jeden Mittwoch und Donnerstag von 08:00 - 12:00 halten wir uns die Zeit frei, um gemeinsam an 2PLAN zu arbeiten. Ausserdem findet jeden zweiten Mittwoch ein Sprint Review und die Sprint Planung statt.

10.3 Prozessmodell

Da wir Scrum plus aus früheren Vorlesungen kennen, haben wir uns entschieden, diesen Workflow für unsere BA zu verwenden. Scrum plus ist eine Kombination aus Scrum und Unified Process. Vom Unified Process übernehmen wir das Konzept der Phasen Inception, Elaboration, Construction und Transition. Von Scrum übernehmen wir die agile Entwicklung mit Sprints. In jeder Projektphase führen wir Sprints von 2 Wochen durch, die es uns

ermöglichen, effizient zu arbeiten und auf unerwartete Ereignisse zu reagieren. In der Elaboration und Construction Phase haben wir je einen Sprint, der nur eine Woche dauert und als Reserve dient. Die Sprints beginnen und enden immer am Mittwoch.

Phasen

- Inception: 22.02.2021 - 24.02.2021 (2 Tage)
- Elaboration: 24.02.2021 - 31.03.2021 (5 Wochen)
- Construction: 31.03.2021 - 02.06.2021 (9 Wochen)
- Transition: 02.06.2021 - 16.02.2021 (2 Wochen)

Sprints

- Sprint 0: 22.02.2021 - 24.02.2021
- Sprint 1: 24.02.2021 - 10.03.2021
- Sprint 2: 10.03.2021 - 24.03.2021
- Sprint 3: 24.03.2021 - 31.03.2021 - Reserve
- Sprint 4: 31.03.2021 - 14.04.2021
- Sprint 5: 14.04.2021 - 28.04.2021
- Sprint 6: 28.04.2021 - 12.05.2021
- Sprint 7: 12.05.2021 - 26.05.2021
- Sprint 8: 26.05.2021 - 02.06.2021 - Reserve
- Sprint 9: 02.06.2021 - 16.06.2021

10.4 Releases

Insgesamt haben wir vier Releases geplant. An einem Release wird der Code vom Development auf den Masterbranch gemergt.

Die Releases finden immer zeitgleich zu den Meilensteinen in der Construction Phase statt.

10.5 Meilensteine

Bezeichnung	Datum	Beschreibung
End of Inception	24.02.2021	Kick-Off Meeting wurde durchgeführt und wir haben alle wichtigen Informationen erhalten.
End of Elaboration	31.03.2021	Projektplan definiert, Anforderungsspezifikation, Domainmodel, Datenmodel und Wireframes erstellt, Architektur festgelegt, Toolchain aufgesetzt, Prototyp implementiert und alle wichtigen Risiken behoben.
Core Construction	14.04.2021	API definiert, Views für CRUD Vorhaben, Vorgang, Eigenschaft, Ressourcen und Kalenderansicht implementiert.
Alpha Release (MVP)	28.04.2021	Vorhaben planen und API-Anbindungen implementiert. In diesem Meilenstein wird das MVP fertiggestellt.
Beta Release	12.05.2021	Ressourcen planen, Optionale Validierung, Meldungen erstellen, akzeptieren und überall visuell kennzeichnen, Ressourcenauswahl mit Zusatzinformationen, Quittierung implementiert. Cognitive Walkthrough mit User Experience Designerin durchführen.
Final Release	02.06.2021	UI-Optimierungen, Lock Past und E2E Tests implementieren. Performance Test ausführen und NPM Package generieren.
Final Submission	16.06.2021	Schlussdokumentation erstellt

Tabelle 10.1: Meilensteine

10.5.1 Sprint - Meilenstein Mapping

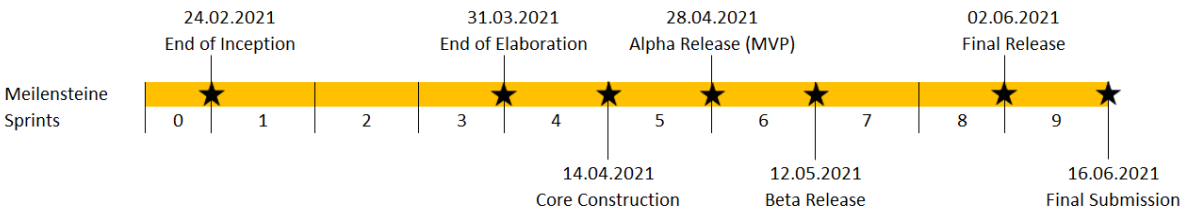


Abbildung 10.1: Sprint - Meilenstein Mapping

10.6 Projektplan

Retrospektiv betrachtet haben wir folgenden Projektplan verfolgt.

2PLAN	Projekt Start	KW	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	22.02.2021	Sprint Start Datum	22. Feb	03. März	10. März	17. Mar	24. Mär	31. Mär	7. Apr	14. Apr	21. Apr	28. Apr	5. Mai	12. Mai	19. Mai	26. Mai	2. Jun	9. Jun
Task		**Datum**																
Inception																		
Kick-Off																		
End of Inception		**24.02.2021**	★															
Elaboration																		
Projektplan																		
Scope Definition																		
Use Cases																		
NFR																		
Domain Model																		
Zustandsdiagramm																		
GUI Mockups																		
Risikoanalyse																		
Datenmodel																		
CI / CD																		
Architekturlayer																		
Prototyp																		
E2E Tests ausprobieren																		
End of Elaboration		**31.03.2021**					★											
Construction																		
API definieren																		
CRUD Vorhaben																		
Kalenderansicht																		
CRUD Vorgang																		
CRUD Eigenschaft																		
CRUD Ressourcen																		
Core Construction		**14.04.2021**							★									
API redefinieren																		
API-Anbindung und Speicherlogik																		
Vorhaben planen																		
Alpha Release (MVP)		**28.04.2021**									★							
Dokumentation Review																		
Mengengerüst																		
Ressourcen planen																		
Optionale Validierung - Szenario 1																		
Messaging																		
Quittierung																		
Ressource Selection mit Zusatzinformationen																		
Cognitive Walkthrough																		
Beta Release		**12.05.2021**											★					
UI-Optimierungen / Bugs																		
E2E																		
Lock Past Implementierung																		
NPM Package																		
Performance Test																		
Final Release		**02.06.2021**														★		
Transition																		
Dokumentation																		
Final Submission		**16.6.2021**																★

Abbildung 10.2: Projektplan

10.6.1 Offene Punkte

Da für uns beide Angular eine neue Programmiersprache war und es zu Problemen bei der API-Architektur kam (siehe Kapitel 1.5.1), verloren wir viel Zeit und wir merkten, dass wir nicht alle Funktionen fertigstellen können. Aus diesem Grund führten wir mit dem Auftraggeber während des Beta Release eine Priorisierung der Arbeiten durch. Folgende Punkte konnten wir leider nicht mehr fertigstellen:

- Ressourcen, die nicht komplett durch Vorgänge ausgelastet sind, werden als verfügbar markiert.

- Auf einem Vorgang können Vorgänger definiert werden. Diese Vorgänger werden aktuell aber nirgends beachtet. Beim Speichern müsste geprüft werden, ob ein Vorgänger auch wirklich vor dem aktuellen Vorgang abgeschlossen wird.
- Die Szenarien 2 - 5, 7 und 8 von der optionalen Validierung (siehe Kapitel 2.4) wurden nicht implementiert. Jedoch wurde die optionale Validierung so implementiert, dass weitere Szenarien einfach eingebaut werden können.
- Der PDF Export von einem Einsatzplan und der Projektplanung fehlt.
- Es fehlt eine Gesamtansicht in Form eines Kalenders, wo der Planer alle Allokationen von allen Ressourcen über alle Vorhaben sieht. Dies sollte aber relativ einfach implementiert werden können, da dafür die Dienstplan-Komponente wiederverwendet werden kann.
- Einbau der Rollen in der Demo-Applikation sodass eine Ressource nur ihren Dienstplan sieht und ein Planer nur die Vorhaben, die ihm zugewiesen wurden.

Für alle offenen Punkte haben wir Backlog Items für 2BIT auf Azure DevOps [9] erstellt.

10.7 Software Entwicklungsprozess

Die Infrastruktur für den Entwicklungsprozess haben wir von 2BIT erhalten. Ausserdem hat 2BIT für uns auf Azure DevOps [9] ein Repository vorbereitet mit einem Grundgerüst der Demo-Applikation. In dieser Demo-Applikation, binden wir unsere Komponenten als Projekt Library ein. Am Schluss des Projektes wird dann diese Projekt Library über NPM [21] zur Verfügung gestellt.

10.7.1 CI / CD

Von 2BIT wurde am Anfang auch bereits eine Pipeline aufbereitet, welche das Backend und die Demo-Applikation buildet und auf einen Server kopiert. Der Code der Demo-Applikation wird dabei mit ESLint [12] überprüft. Da diese Pipeline nur für den ”Master” und ”Development” Branch vorgesehen ist, implementierten wir noch eine zweite Pipeline, die für die Implementations-Branches vorgesehen ist. Diese zweite Pipeline führt die gleichen Schritte wie die erste Pipeline aus, nur werden das Backend und die Demo-Applikation nicht auf einen Server kopiert. Abschliessend fügten wir beiden Pipelines noch die Schritte für den Build der Library hinzu. Zusätzlich dazu wird in der Pipeline für den ”Master” und ”Development” Branch der Publish der Library auf das NPM Repository durchgeführt.

10.7.2 Zeiterfassung

Wir haben zuerst versucht, die Zeiterfassung über Azure DevOps mit TMetric [30] durchzuführen. Jedoch haben wir es nicht hinbekommen, eine geeignete Zeitauswertung zu machen. Um nicht zu viel Zeit zu verlieren, haben wir uns dafür entschieden, die Zeit wie bisher in allen Projekten über OST Gitlab [33] zu rapportieren. Nach Absprache mit Raphael Ritter werden wir auch die Issues über OST Gitlab [33] pflegen.
Mit dem Tool GTT (GitLab Time Tracker) können wir die Zeiten auswerten.

10.8 Qualitätsmassnahmen

10.8.1 Dokumentation

Das Hauptdokument wird mit dem LaTeX Editor Overleaf [22] geschrieben. Zusätzlich dazu gibt es eine technische Dokumentation auf Englisch mit einer Anleitung, wie man die Library

einbindet und das Tool anwendet. Diese Dokumentation wird als Wiki direkt in Azure DevOps [9] erfasst.

Nach Abschluss einer Änderung an einem Dokument weist man dem anderen Projektmitglied den Issue zu. Anschliessend überprüft dieser die Änderungen und korrigiert Schreibfehler. Falls es inhaltliche Fehler gibt, schreibt man einen Kommentar in den Issue und es werden nochmals Anpassungen vorgenommen.

10.8.2 Branches

Im Code Repository gibt es die zwei Branches "Master" und "Dev". Pro Feature wird ein weiterer Branch auf Basis des Development Branches erstellt. Sobald man mit einem Issue fertig ist, wird über ein Merge Request die Änderung auf den Development Branch geschrieben. Nur wenn ein Release ansteht oder ein Hotfix implementiert wurde, wird ein Merge Request vom Development auf den Master Branch ausgeführt.

Die Branches werden wie folgt benannt:

- Format: <type>-<issue number>-<description>
- Beispiel: feature-811-implement_message_component
- Beispiel: bugfix-812-implement_message_types

10.8.3 Code Review

Bei jedem Merge Request auf den Development Branch wird ein Code Review durchgeführt. Ziel davon ist die Codequalität zu steigern, Duplicated Code zu vermindern und den Wissensaustausch zu fördern.

Der Pull Request wird im Azure DevOps[9] erfasst und zusammen mit dem Issue dem anderen Projektmitglied zugewiesen. Wenn alles in Ordnung ist, wird der Pull Request akzeptiert und die Änderungen werden auf den Development Branch gemerged. Falls etwas nicht in Ordnung ist, werden im Pull Request an den entsprechenden Codesstellen Kommentare erfasst. Anschliessend weist man den Issue wieder dem anderen Projektmitglied zu und informiert ihn, dass der Request noch nicht akzeptiert wurde.

10.8.4 Code Qualität

Bei jedem GitLab CI-Lauf wird der Code mit dem Code-Analyse-Tool ESLint [12] nach dem Standart von 2BIT überprüft. ESLint [12] hilft problematischen Code und Code, der sich nicht an Guidelines hält, zu finden.
Zusätzlich zu ESLint [12] wird bei jedem Gitlab CI-Lauf Prettier [23] ausgeführt. Prettier [23] definiert, wie der Code formatiert werden soll.

10.8.5 Definition of Done

Die folgenden Kriterien müssen erfüllt sein, bevor ein Issue als abgeschlossen betrachtet wird

- CI-Pipeline ist erfolgreich durchgelaufen
- Alle Tests laufen fehlerfrei
- Ein Review wurde durchgeführt
- Dokumentation nachgeführt

10.8.6 Code Freeze

Zwei Wochen vor End of Construction, also am 19.05.2021, wird ein Feature Freeze angesetzt. Danach sind bis zum Code Freeze am 02.06.2021 nur noch Bug-Fixes und Verbesserungen erlaubt. Falls anschliessend noch ein Problem entdeckt wird, muss miteinander abgewogen werden, ob es genug wichtig ist um zu korrigieren. Falls ja, wird das Problem miteinander analysiert und behoben.

10.9 MVP

Das MVP umfasst die nachfolgende Kernfeatures, die bis zum Abschluss des Meilensteins "Alpha Release" entwickelt werden.

- Die Stammdatenverwaltung für den Administrator wird implementiert. Vorhaben, Vorgänge, Ressourcen und Eigenschaften können erstellt und bearbeitet werden.
- Der Planer kann Vorhaben planen und neue Vorgänge erstellen. Die Planung wird in einer Vorhabensansicht auf Wochenbasis gemacht.
- Ein User kann die für ihn geplanten Vorhaben in einer Kalenderansicht einsehen.
- Ressourcen können einem Vorgang zugewiesen werden.
- Die Lokalisation für die verschiedenen Sprachen wird implementiert.

10.10 Non MVP

Die nachfolgenden Features werden nach Abschluss des MVP implementiert.

- Beim Zuweisen einer Ressource zu einem Vorgang werden optionale Validierungen durchgeführt. Bei der Allokation werden Fehler / Warnungen generiert. Diese Meldungen können akzeptiert werden, sodass sie nicht mehr angezeigt werden.
- Nicht komplette Vorgänge, freie Ressourcen und Konflikte werden visuell dargestellt
- Vergangene Pläne können nicht verändert werden
- Feiertage werden ausgelesen und in der Planungsansicht visuell gekennzeichnet
- In der Planungsansicht kann zwischen Gesamt-, Vorhabens- und Ressourcenansicht hin und her gewechselt werden.
- Die Planung kann auf Tages-, Wochen- oder Monatsbasis durchgeführt werden.
- Der Planer, Admin und der User können verschiedene Pläne als PDF exportieren.
- Der Planer kann einstellen, ob ein Vorgang quittiert werden muss oder nicht. Jede Ressource die für eine quittierbare Aktivität eingeplant wird, kann die Quittierung vornehmen.
- Das Login wird über ein Active Directory durchgeführt. Die verschiedenen Componenten werden durch Benutzerrollen abgegrenzt.

Teil III

Administrative Anhänge

Anhang A

Risikoanalyse

A.1 Risiken

Nachfolgend ist eine Übersicht über alle evaluierten Risiken zu finden. Die Spalten für die Eintrittswahrscheinlichkeit wurden mit “EW“ bezeichnet.

Id	Titel	Beschreibung	Massnahmen	Reserve Rückstellungen (h)	Impact	Label	Eingetroffen	EW (%)	EW M2 (%)	EW M3 (%)	EW M4 (%)	EW M5 (%)	EW M6 (%)
1	Testing Komplikationen	Die Angular Testing Frameworks sind Neuland für das Team. Das Team wird das Karma Framework zusammen mit E2E Tests eingesetzt, da diese als Angular Standard angesehen werden.	Im Prototyp werden bereits E2E Tests und Karma Tests implementiert, um ein Verständnis für den Ablauf zu bekommen.	8	2	1 Testing Komplikationen	Ja	25	15	15	15	15	5
2	Data Binding	Angular ist eine neue Technologie für das Team. Das Konzept mit Input und Output gehört dabei auch zum Unbekannten. Dieses Konzept wird jedoch eingesetzt, um Daten aus der Demo App der Library zur Verfügung zu stellen und umgekehrt.	Das Team hat sich bereits vor dem Beginn der Arbeit mit dem Hero Tutorial beschäftigt. Weiterhin wird im Prototyp das Konzept ausprobiert.	0	1	2 Data Binding	Nein	15	5	5	5	5	5
3	Schwierigkeiten CI Pipeline	Das Team hat noch nie mit Azure DevOps gearbeitet. Das Aufsetzen der CI/CD Pipeline stellt deshalb eine Herausforderung dar, die Verzögerungen auslösen könnte.		4	2	3 Schwierigkeiten CI Pipeline	Ja	45	5	5	5	5	5
4	Workload Use Cases	Der Workload der Use Cases ist höher als für den jeweiligen Use Case geschätzt. Dies führt zu Verzögerungen im Meilenstein und schlussendlich im gesamten Projektplan.	In der Planung wurde dies berücksichtigt und der Umfang eines Sprints entsprechend geplant.	20	1	4 Workload Use Cases	Nein	15	15	15	15	15	5
5	Schnittstellen Komplikationen	Das Team hat Schwierigkeiten die Applikation an die externen Schnittstellen anzubinden.	Um sicherzustellen, dass es keine unvorhergesehenen Schwierigkeiten bei der Anbindung an die Schnittstellen gibt, wird bereits in der Elaboration Phase ein Prototyp erstellt, der alle Schnittstellen implementiert.	8	2	5 Schnittstellen Komplikationen	Nein	35	5	5	5	5	5
6	OpenSource Library	Für die gewählte Open Source Library der Gantt/Kalender Ansicht müssen Änderungen am Design gemacht werden. Die Änderungen stellen sich als äusserst mühselig heraus und kann zu Verzögerungen führen.	Die Libraries wird im Prototyp evaluiert.	8	3	6 OpenSource Library	Ja	25	15	15	5	5	5
7	Schlechte Usability	Das Produkt ist vom End User schlecht benutzbar. Die Bedienung des Produktes ist nicht intuitiv.	Bereits in der Elaboration Phase werden klickbare GUI Mockups mit Balsamiq erstellt. Diese werden dem Auftraggeber zur Begutachtung gegeben. Nachdem die Applikation steht, werden wir ausserdem noch professionelle Beratung von einer Design Expertin im User Experience Bereich einholen.	16	4	7 Schlechte Usability	Nein	25	15	15	15	5	5
8	Ausfall eines Arbeitsgerätes	Das Arbeitsgerät eines Teammitglied fällt aus und das Teammitglied kann nicht weiterarbeiten.	Ein Ersatzlaptop steht bereit, welcher gleich zur Verfügung gestellt werden kann.	0	1	8 Ausfall eines Arbeitsgerätes	Nein	5	5	5	5	5	5
9	Erkrankung eines Teammitglied	Ein Teammitglied fällt wegen einer Krankheit (Covid-19) für eine unbestimmte Zeit aus und kann die zugewiesene Arbeit erst nach Genesung abschliessen.	Die Teammitglieder halten sich streng an die Vorgaben des BAG, um Infektion zu vermeiden.	40	3	9 Erkrankung eines Teammitglied	Nein	15	15	15	15	15	15
10	API Architektur	Aufgrund von verschiedenen Ansichten und Funkstille zwischen unserem Betreuer und unserem Kunden gab es eine neu-evaluierung der API. Das Ergebnis dieser neu-evaluierung gibt zusätzlichen Aufwand und verzögert die Weiterentwicklung, da die API noch umgebaut werden muss.	Der Aufbau der API wird mit Betreuer und Kunde zusammen nochmals besprochen. Die definitive Ausarbeitung findet mit einem Entwickler von 2Bit statt.	0	3	10 API Architektur	Ja			45	35	25	15
Total:				104									
Total eingetroffener Schaden (h):				62									

Abbildung A.1: Risikoliste

A.2 Risiko Diagramm

A.2.1 Erstschätzung

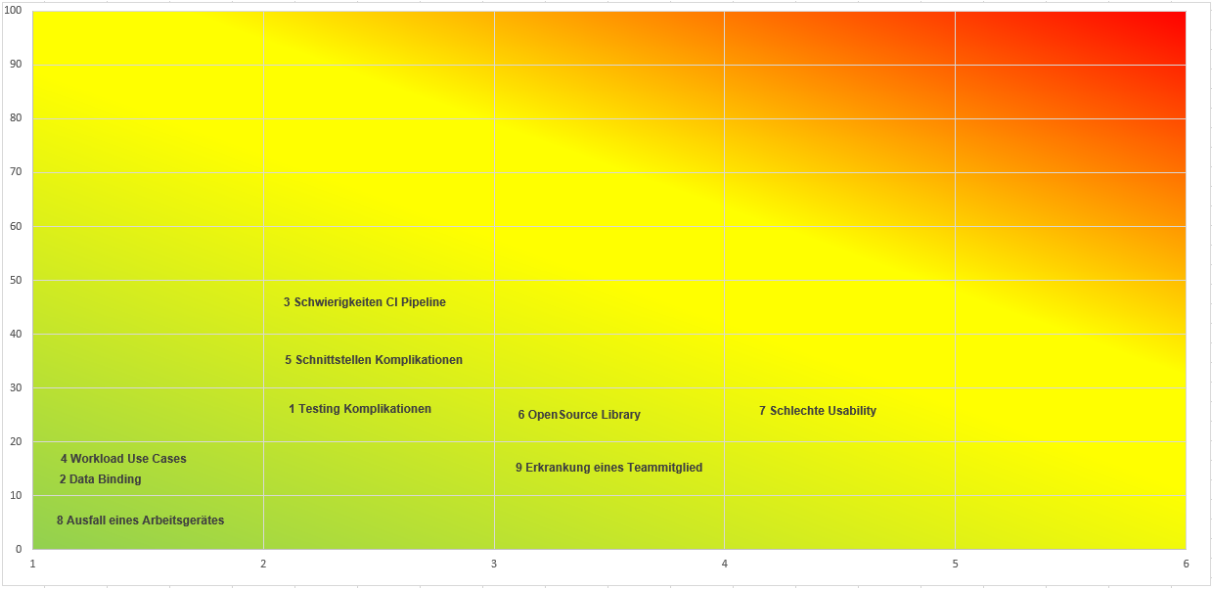


Abbildung A.2: Risiko Diagramm Erstschätzung

A.2.2 Schätzung M2 - End of Elaboration

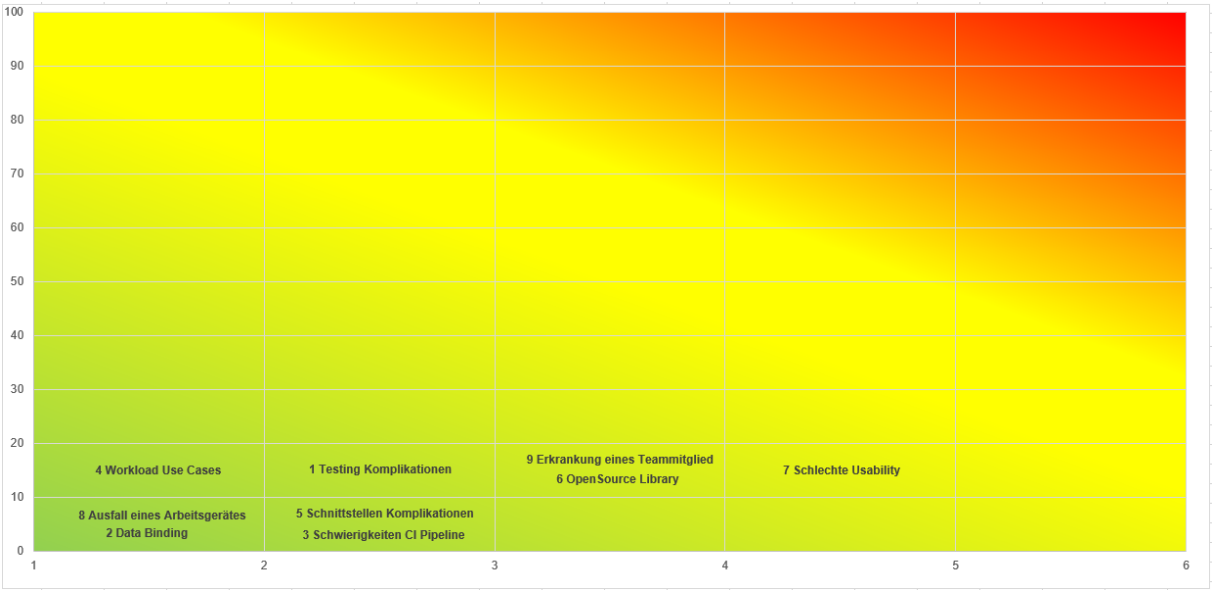


Abbildung A.3: Risiko Diagramm End of Elaboration

Durch die Implementierung des Prototypen konnten wir bereits beim Meilenstein "End of Elaboration" die Risiken 1, 2, 3, 5 und 6 minimieren. Das Risiko mit Nummer 7 konnten wir durch die erstellten Wireframes und durch die Abnahme des Kunden minimieren.

A.2.3 Schätzung M3 - Core Construction

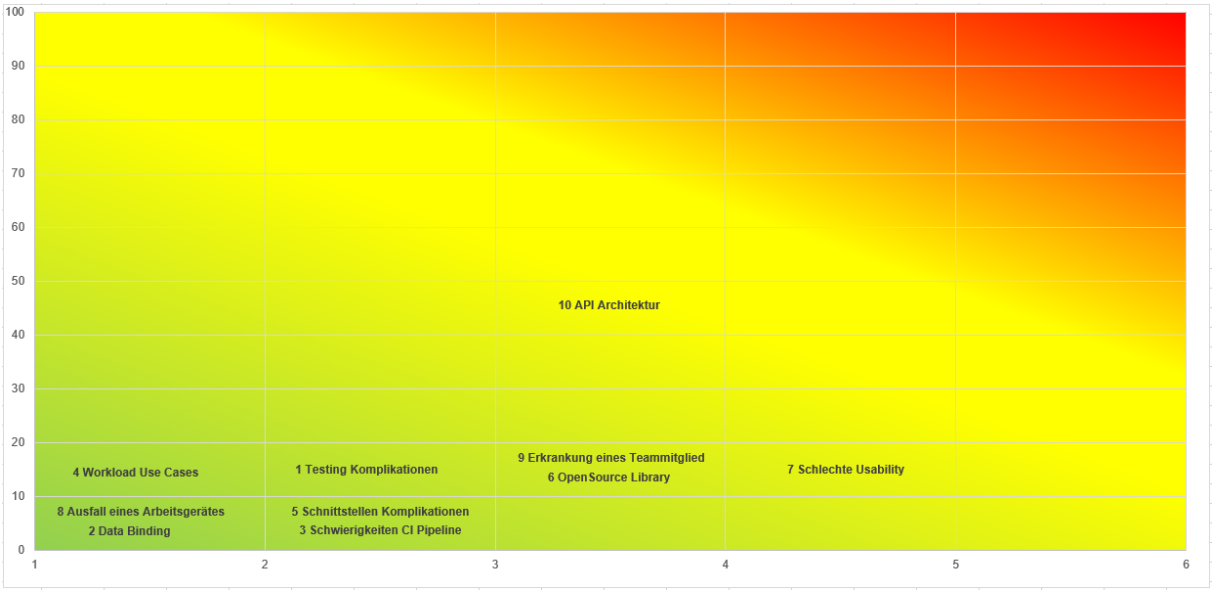


Abbildung A.4: Risiko Diagramm Core Construction

In diesem Meilenstein kam ein neues Risiko dazu, das uns im Vorhinein nicht bewusst war und uns am meisten Probleme verursacht hat. Das Problem entstand durch zwei verschiedene Ansichten bezüglich der API-Architektur. Für die API-Architektur haben wir zuerst die Meinung unseres Betreuers eingeholt, der den Best Practice Ansatz (Graph-Architektur) vertrat. Da auch wir der Meinung waren, dass dies der schönste Ansatz ist, schrieben wir eine API-Definition, die 2BIT bereits anfing zu implementieren. Bei der Präsentation der Architektur gegenüber dem Auftraggeber kam jedoch heraus, dass sie in der Firma jeweils das API mit der Flow-Architektur implementieren. Aus diesem Grund wurde ein Meeting mit allen Parteien gehalten, um zu besprechen, wie wir fortfahren sollten. In diesem Meeting entschieden wir uns dazu, die Variante des Auftraggebers zu implementieren. Daraufhin mussten wir die API neu definieren, was uns Zeit kostete. Zusätzlich ging es etwas länger, bis 2BIT uns die API zur Verfügung stellen konnte.

A.2.4 Schätzung M4 - Alpha Release

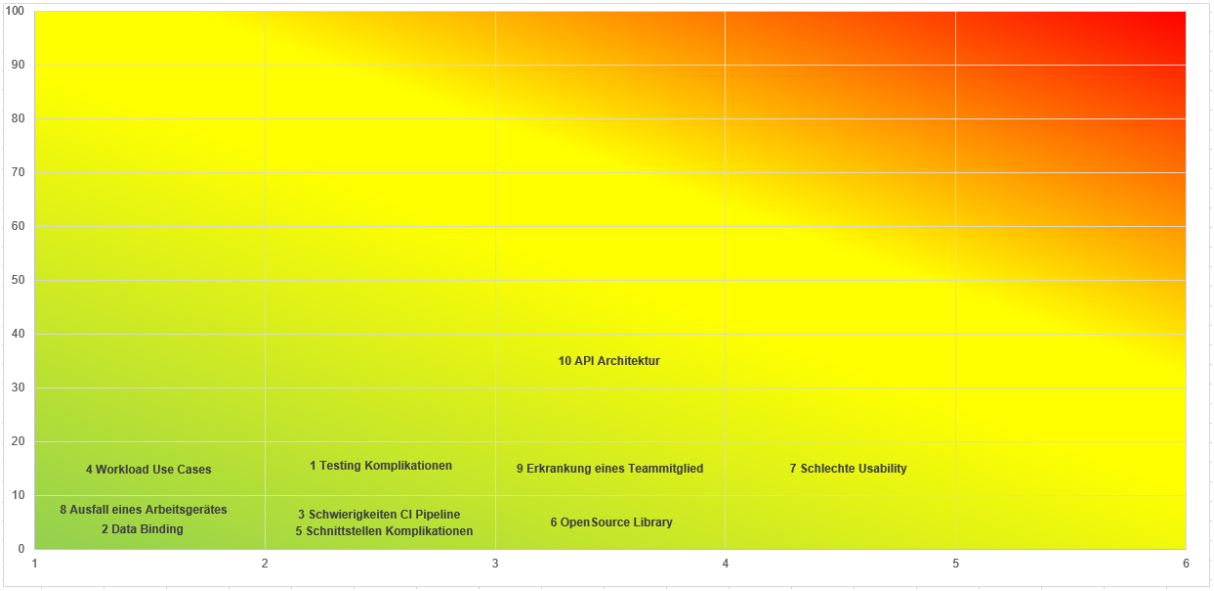


Abbildung A.5: Risiko Diagramm Alpha Release

Ab diesem Meilenstein ist das einzige bestehende Risiko, das noch nicht minimiert wurde, Nummer 10. Durch stetige Verbesserungen und Anpassungen an der API konnten wir jedoch das Risiko von Meilenstein zu Meilenstein minimieren.

A.2.5 Schätzung M5 - Beta Release

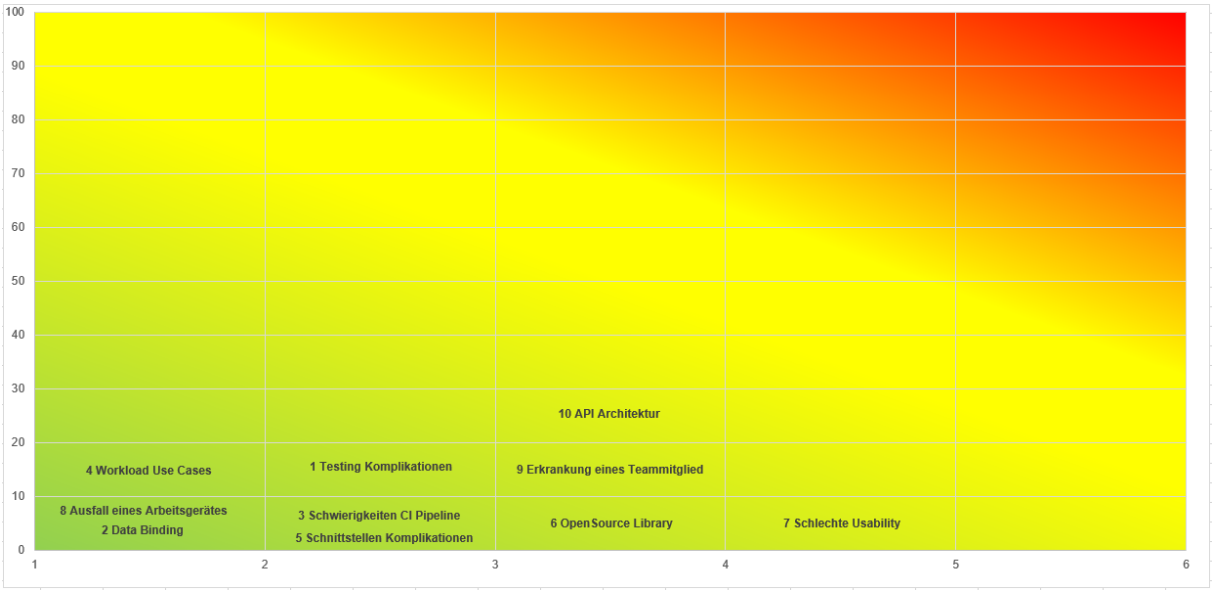


Abbildung A.6: Risiko Diagramm Beta Release

A.2.6 Schätzung M6 - Final Release

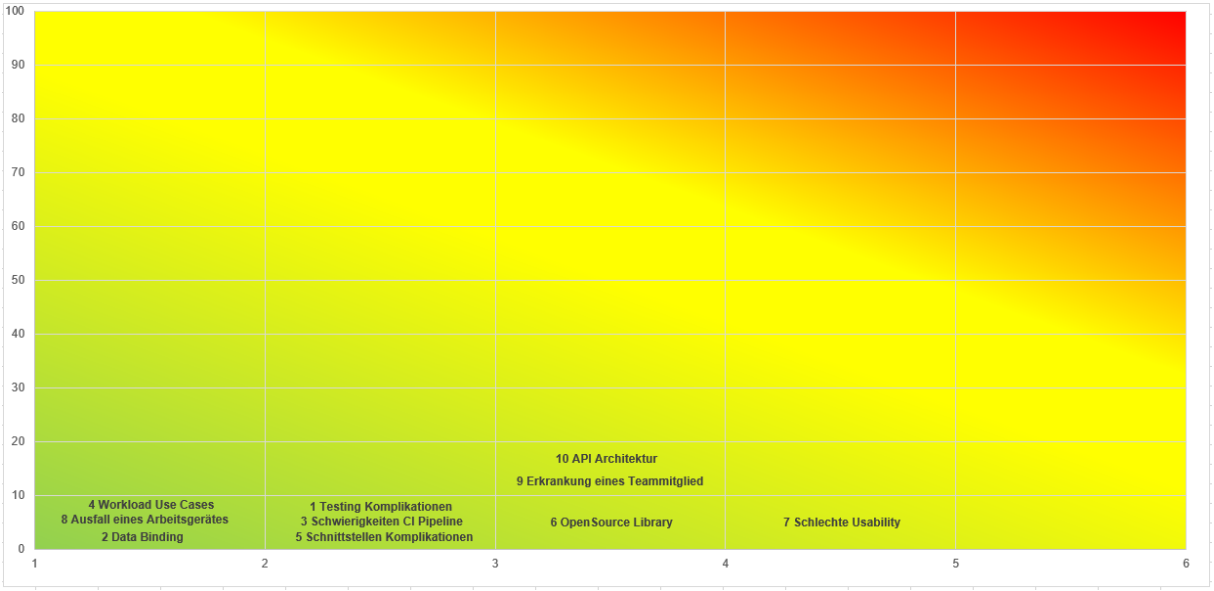


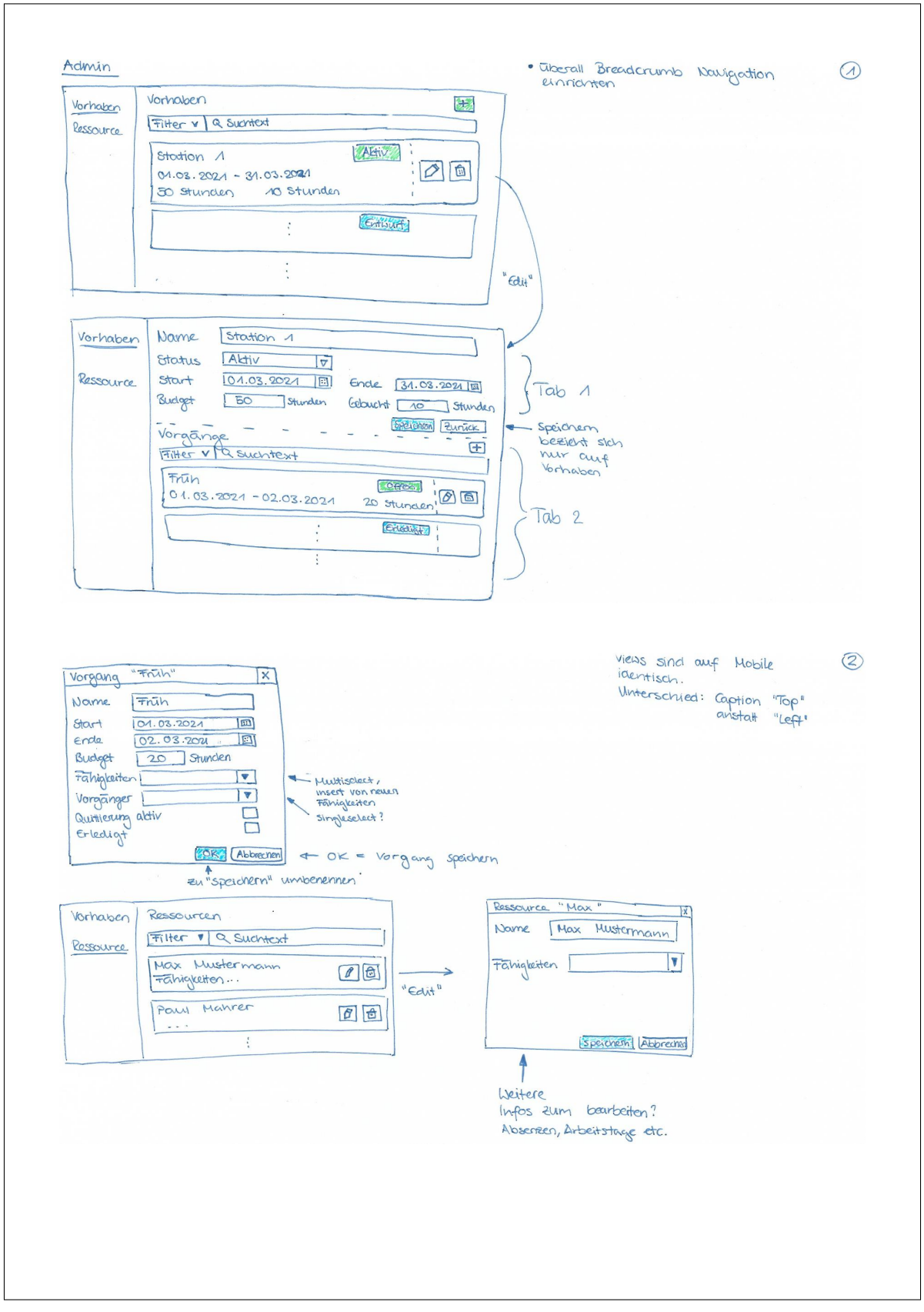
Abbildung A.7: Risiko Diagramm Final Release

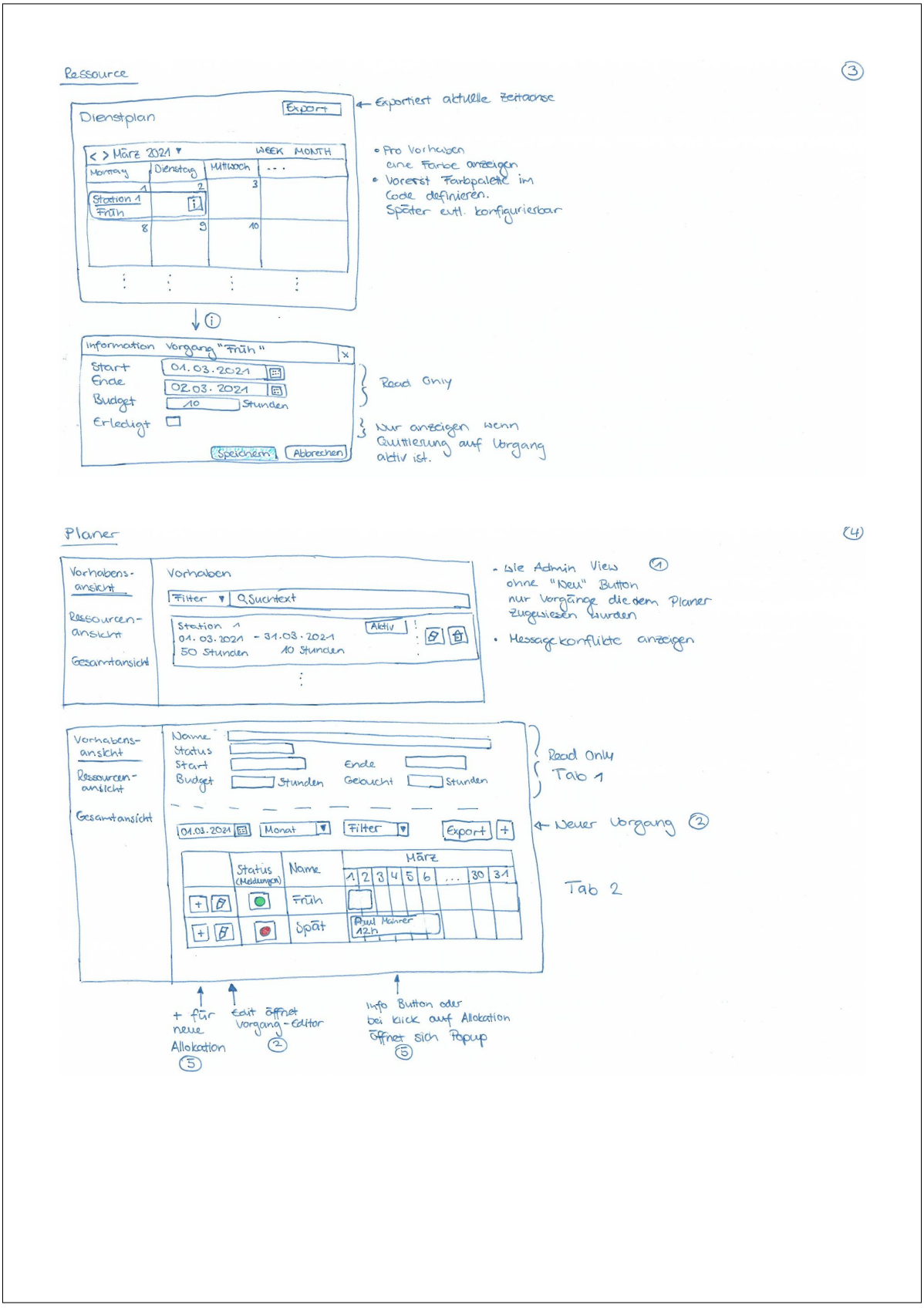
Anhang B

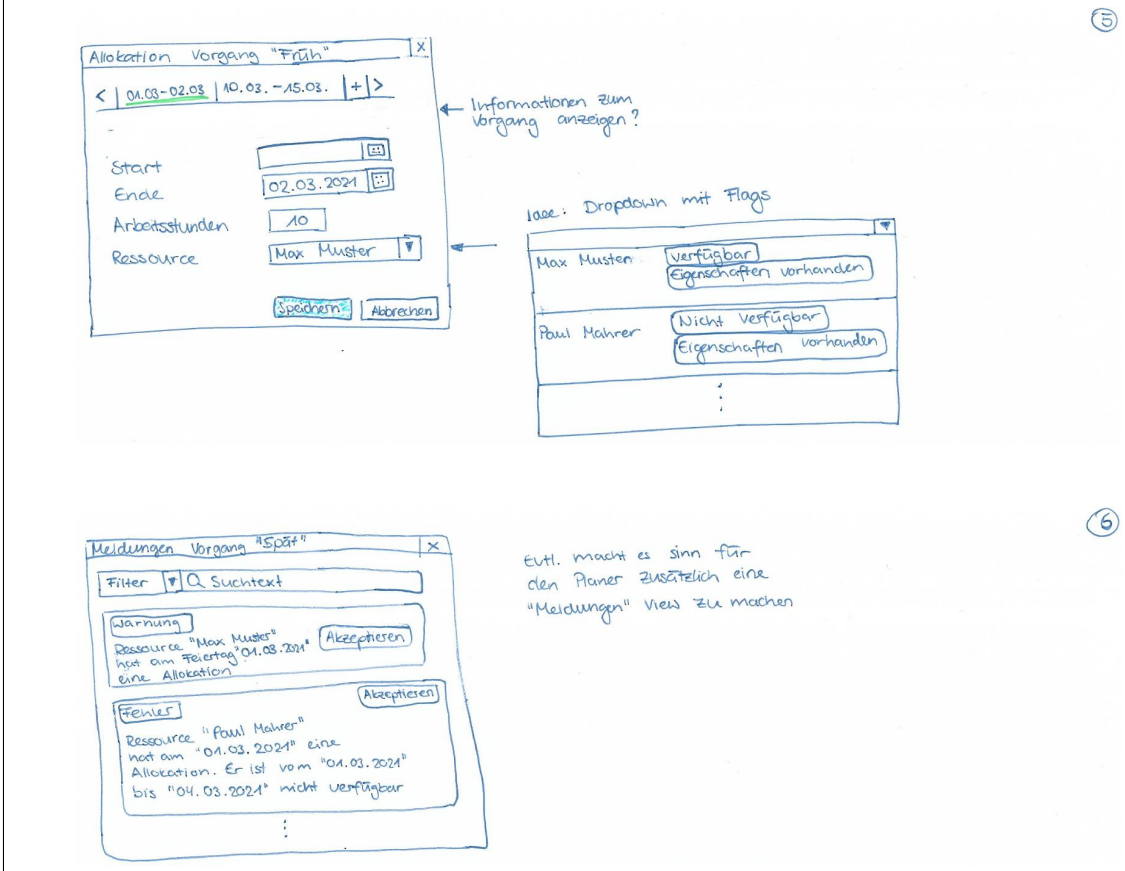
Wireframes

B.1 Handskizzen

Für die Ausarbeitung der Wireframes haben wir zuerst Entwürfe von Hand für die Desktopansicht erstellt.

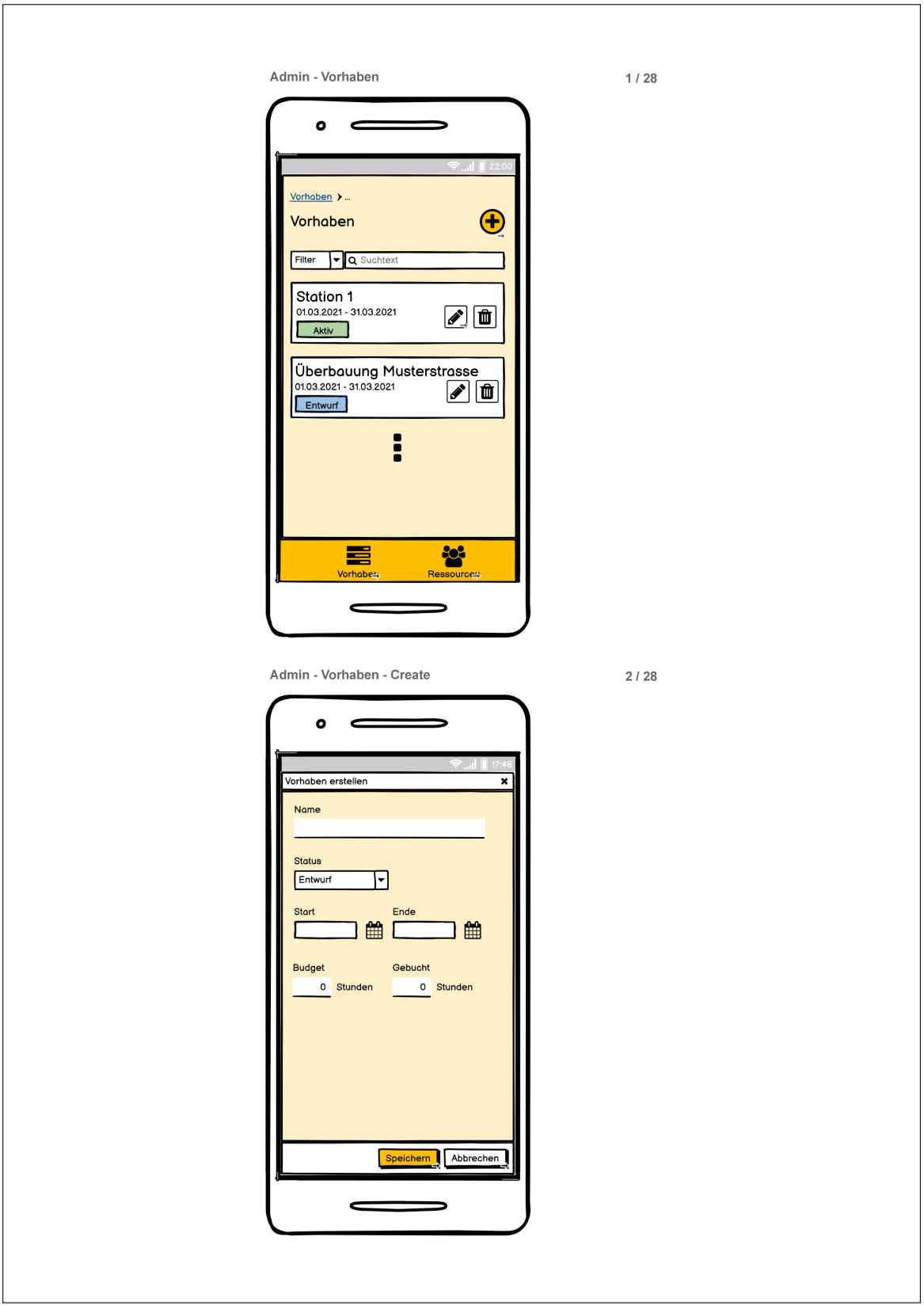

Admin
• Überall Breadcrumb Navigation einrichten
1
Vorhaben
Ressource
Vorhaben
Filter
Suchtext
Station 1
Aktiv
01.03.2021 - 31.03.2021
30 Stunden
10 Stunden
Entwurf
"Edit"
Vorhaben
Ressource
Name
Station 1
Status
Aktiv
Start
01.03.2021
Ende
31.03.2021
Budget
50
Stunden
Gebucht
10
Stunden
Übernehmen
Zurück
Tab 1
Speichern bezieht sich nur auf Vorhaben
Vorgänge
Filter
Suchtext
Früh
Offen
01.03.2021 - 02.03.2021
20 Stunden
Erledigt
Tab 2
Views sind auf Mobile identisch.
Unterschied: Caption "Top" anstatt "Left"
2
Vorgang "Früh"
Name
Früh
Start
01.03.2021
Ende
02.03.2021
Budget
20
Stunden
Fähigkeiten
Vorgänger
Quittierung aktiv
Erledigt
OK
Abbrechen
Multiselect, insert von neuen Fähigkeiten
Singleselect?
OK = Vorgang speichern
zu "Speichern" umbenennen
Vorhaben
Ressource
Ressourcen
Filter
Suchtext
Max Mustermann
Fähigkeiten...
Paul Mahrer
...
"Edit"
Ressource "Max"
Name
Max Mustermann
Fähigkeiten
Speichern
Abbrechen
Weitere Infos zum bearbeiten?
Absenzen, Arbeitstage etc.


Ressource
3
Dienstplan
Export
Exportiert aktuelle Zeitachse
< > März 2021
WEEK MONTH
Montag
Dienstag
Mittwoch
Station 1
Früh
• Pro Vorhaben eine Farbe anzeigen
• Vorerst Farbpalette im Code definieren. Später evtl. konfigurierbar
Information Vorgang "Früh"
Start
01.03.2021
Ende
02.03.2021
Budget
10
Stunden
Erledigt
Speichern
Abbrechen
Read Only
Nur anzeigen wenn Quittierung auf Vorgang aktiv ist.
Planer
4
Vorhabens-ansicht
Ressourcen-ansicht
Gesamtansicht
Vorhaben
Filter
Suchtext
Station 1
01.03.2021 - 31.03.2021
50 Stunden
10 Stunden
Aktiv
- wie Admin View (1) ohne "Neu" Button nur Vorgänge die dem Planer zugewiesen wurden
• Message konflikte anzeigen
Name
Status
Start
Ende
Budget
Stunden
Gebucht
Read Only Tab 1
01.03.2021
Monat
Filter
Export
Neuer Vorgang (2)
Status (Meldungen)
Name
März
Früh
Spät
Paul Mahrer 12h
Tab 2
+ für neue Allokation (5)
Edit öffnet Vorgang-Editor (2)
Info Button oder bei Klick auf Allokation öffnet sich Popup (5)

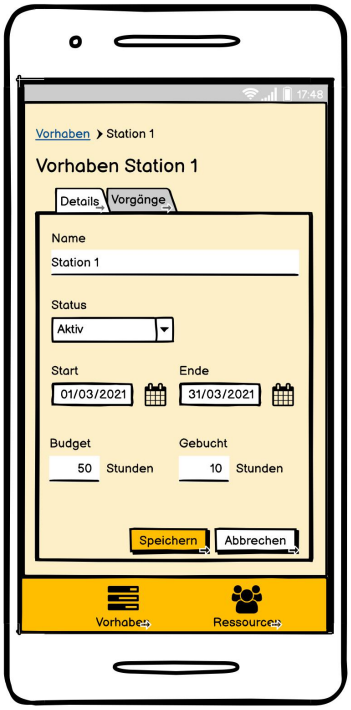

5
Allokation Vorgang "Früh"
< | 01.03-02.03 | 10.03.-15.03. | + | >
Informationen zum Vorgang anzeigen?
Start
Ende
02.03.2021
Arbeitsstunden
10
Ressource
Max Muster
Speichern
Abbrechen
Idee: Dropdown mit Flags
Max Musten
Verfügbar
Eigenschaften vorhanden
Paul Mahrer
Nicht Verfügbar
Eigenschaften vorhanden
6
Meldungen Vorgang "Spät"
Filter
Suchtext
Warnung
Ressource "Max Muster" hat am Feiertag "01.03.2021" eine Allokation
Akzeptieren
Akzeptieren
Fehler
Ressource "Paul Mahrer" hat am "01.03.2021" eine Allokation. Er ist vom "01.03.2021" bis "04.03.2021" nicht verfügbar
Evtl. macht es sinn für den Planer zusätzlich eine "Meldungen" View zu machen

B.2 Balsamiq

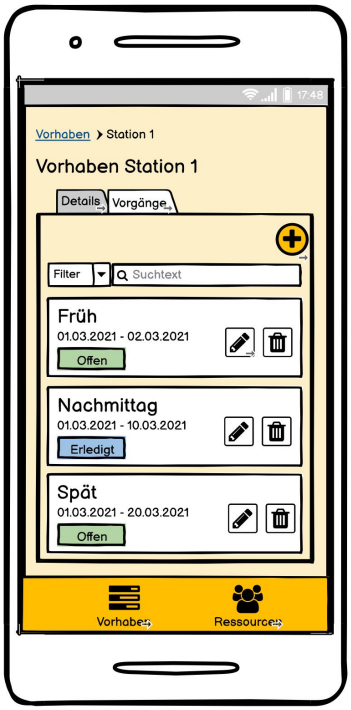
Nachdem wir von Hand erste Skizzen gezeichnet haben, haben wir mithilfe von Balsamiq klickbare Wireframes für die Mobileansicht erstellt. Ausserdem haben wir mit farbigen Hintergründen die Aufteilung der Views zwischen Demo-Applikation (Gelb) und Library (Blau) visualisiert.



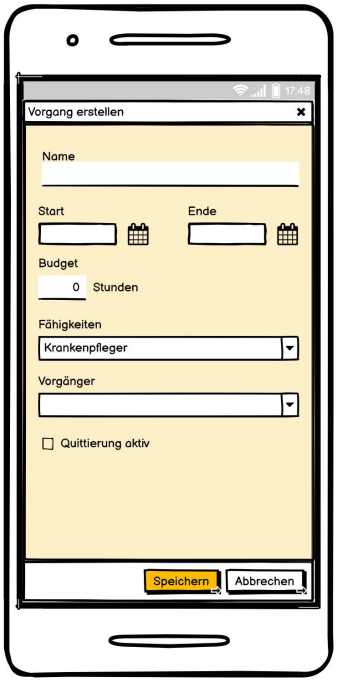
Admin - Vorhaben - Details 3 / 28



Admin - Vorhaben - Vorgänge 4 / 28

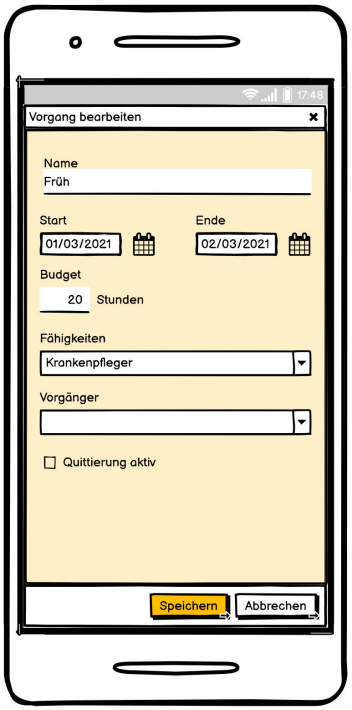


Admin - Vorgang - Create 5 / 28

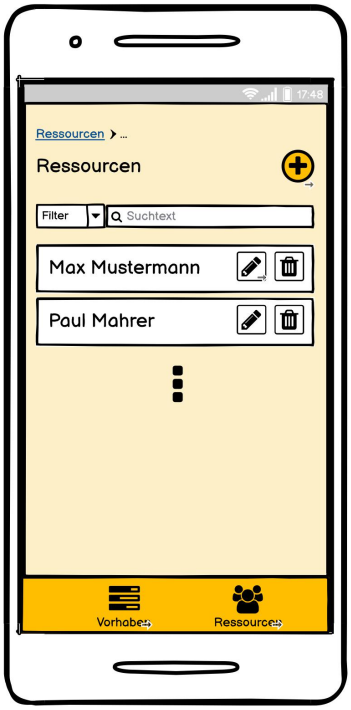


Fähigkeit ist ein Multiselect Combobox.

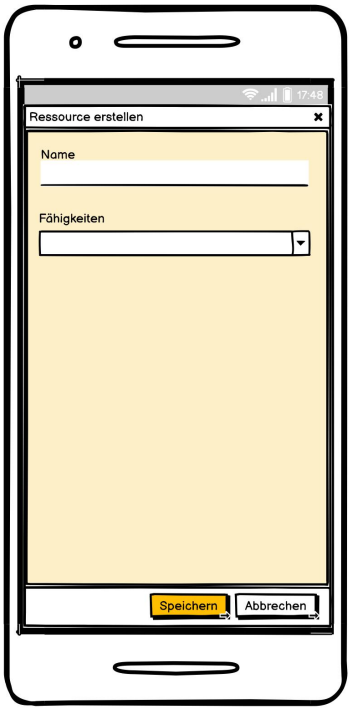
Admin - Vorgang - Edit 6 / 28



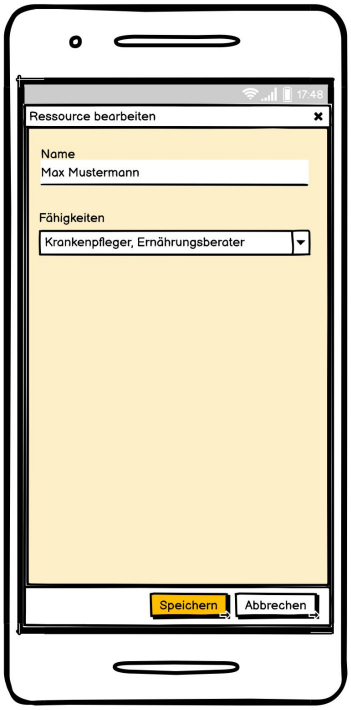
Admin - Ressource 7 / 28



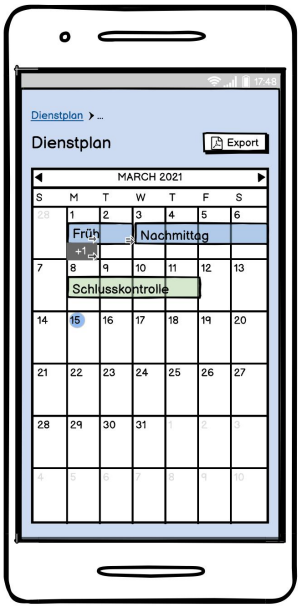
Admin - Ressource - Create 8 / 28



Admin - Ressource - Edit 9 / 28



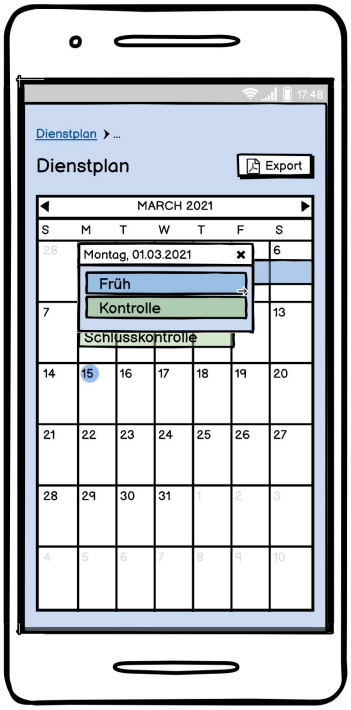
Ressource - Dienstplan 10 / 28



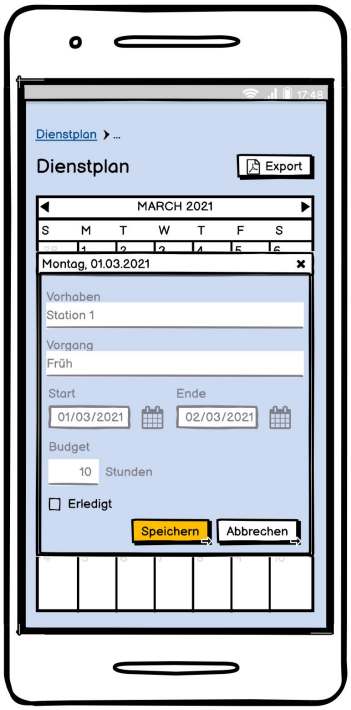
Beschluss Wochenansicht: 07.04.2021
- Orientierung am Google Kalender. Links stehen Stunden und oben das Datum
- jede Stunde wird in zwei Bereich aufgeteilt. Termine mit Minute 0 bis 30 werden im ersten Bereich angezeigt und Termine, die länger gehen werden auch im zweiten Bereich angezeigt.
- Wenn eine Allokation max. über 2 Tage geht, zeigen wir Slots an. So können wir auch Schichtpläne abdecken, bei der eine Person über Nacht arbeitet.
- Wenn eine Allokation über mehr als 2 Tage geht, wird der Slot oben (siehe "Test") angezeigt.

Neben MARCH 2021 wird es ein Dropdown geben um zwischen Wochen und Monatsansicht zu wechseln.
Am Anfang findet die Navigation über die Buttons statt. Zu einem späteren Zeitpunkt soll mit Swipe links/rechts der Monat/Woche gewechselt werden können.

Ressource - Dienstplan - +1 11 / 28

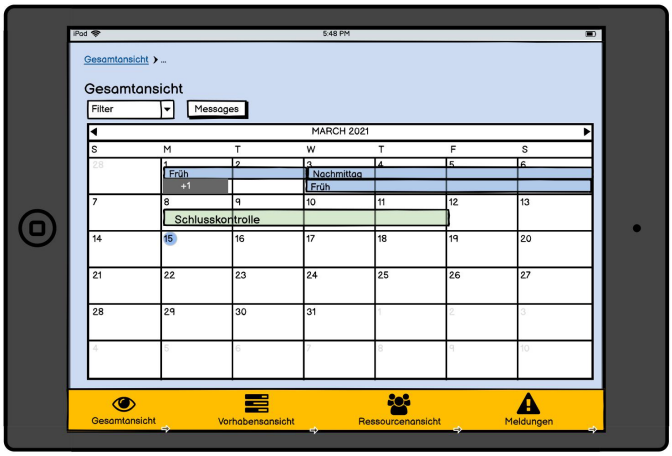


Ressource - Dienstplan - Detail 12 / 28



Planer - Gesamtansicht

13 / 28

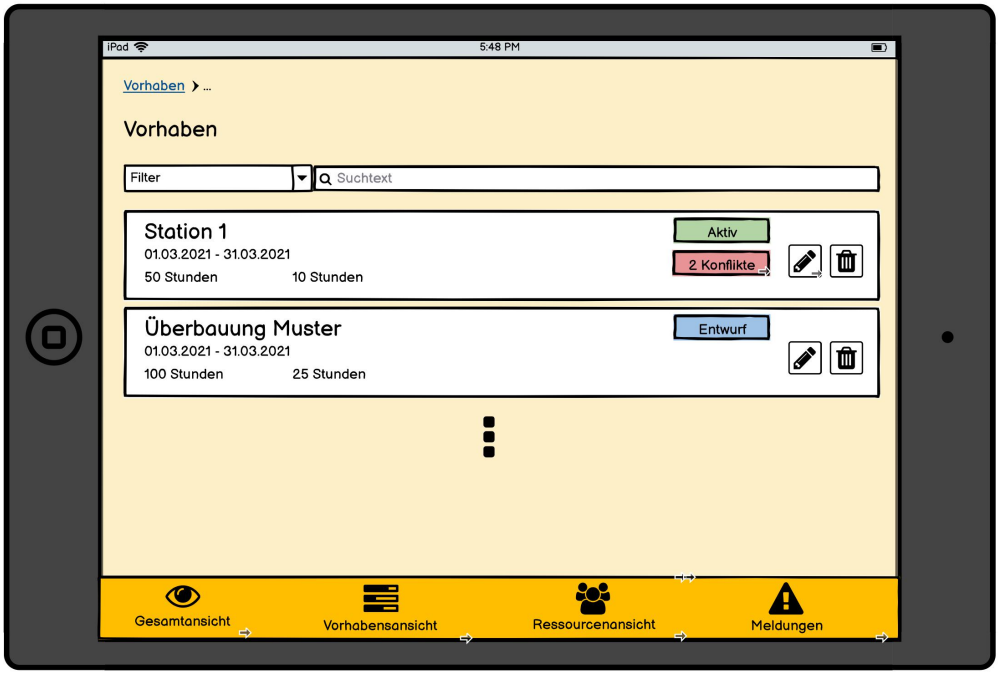


In der Gesamtansicht werden alle Vorgänge von allen Vorhaben angezeigt.
Da dies sehr schnell unübersichtlich wird, gibt es verschiedene Filter.

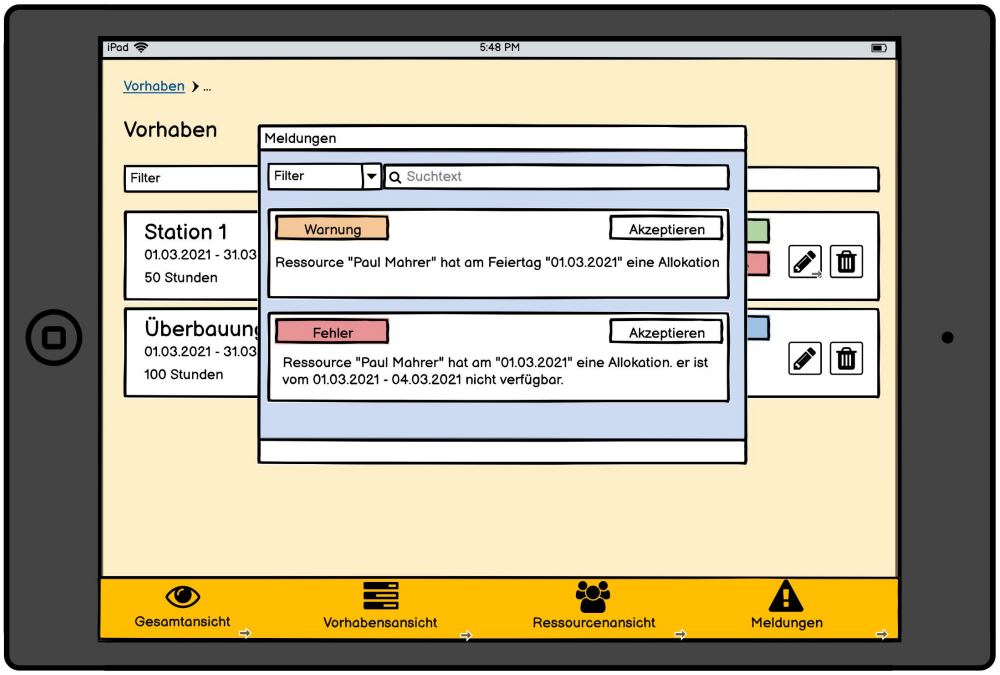
Wie sind in der Gesamtansicht Planungsfehler ersichtlich?
1. Farbliche markierung (Fehler = rot, Warnung = orange, Info = blau)
2. Filter zum nur Vorgänge mit Fehlern anzuzeigen
3. Button um Message Dialog anzuzeigen

Planer - Vorhabensansicht

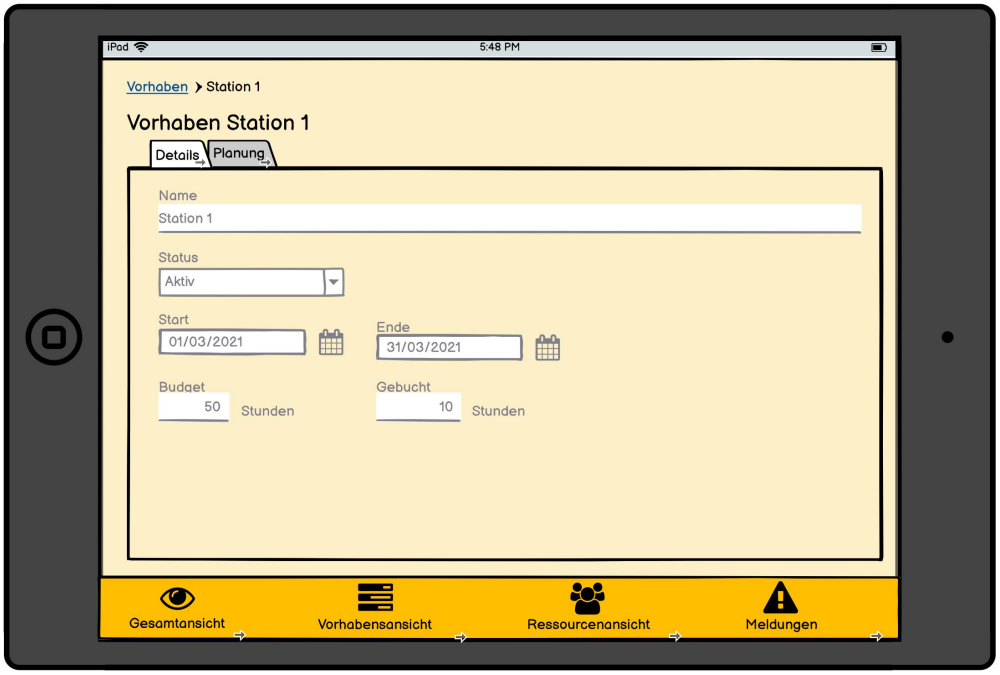
14 / 28



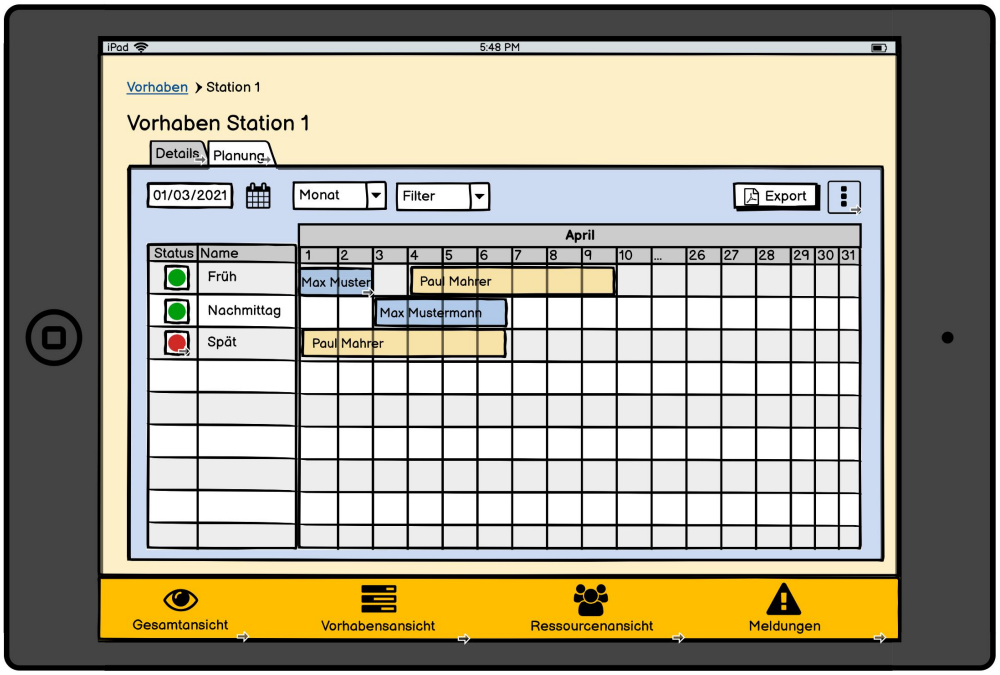
Planer - Vorhabensansicht - Message 15 / 28



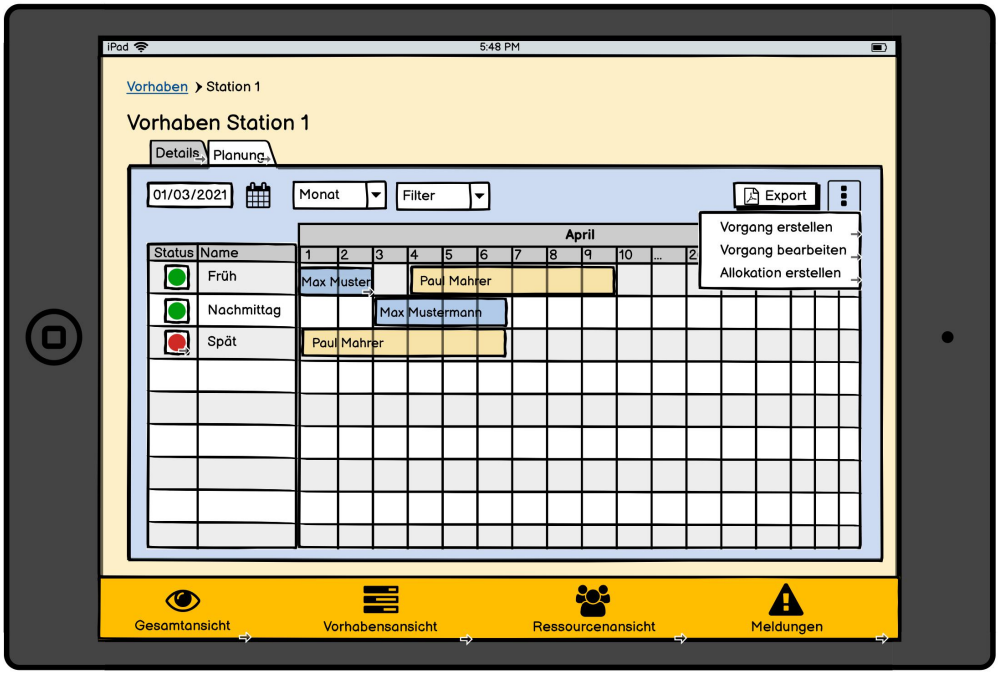
Planer - Vorhabensansicht - Detail 16 / 28



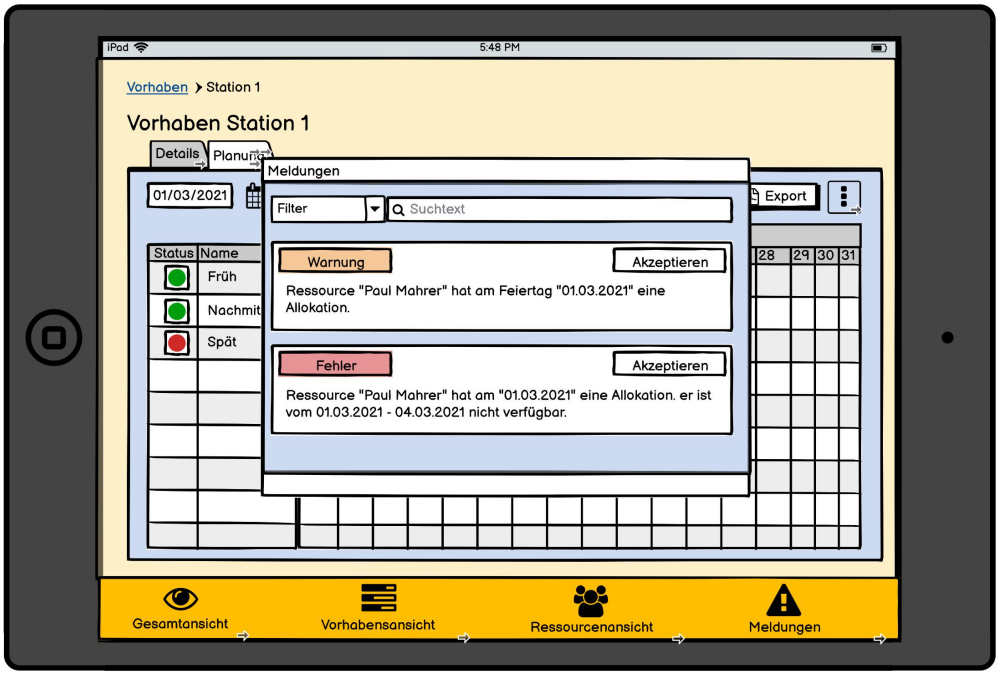
Planer - Vorhabensansicht - Gantt 17 / 28



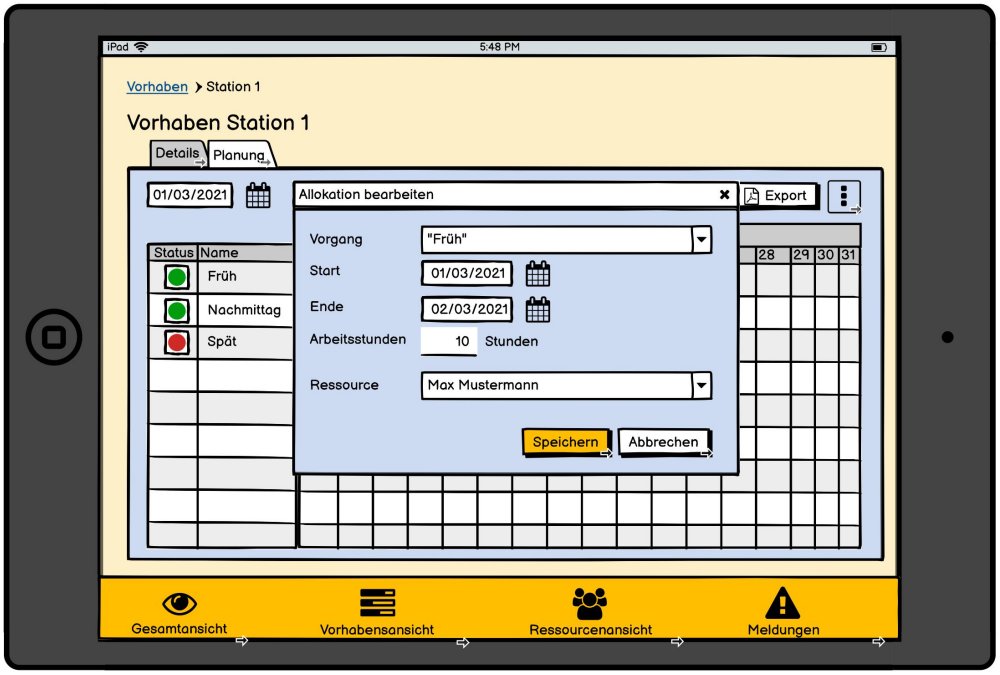
Planer - Vorhabensansicht - Gantt - 2 18 / 28



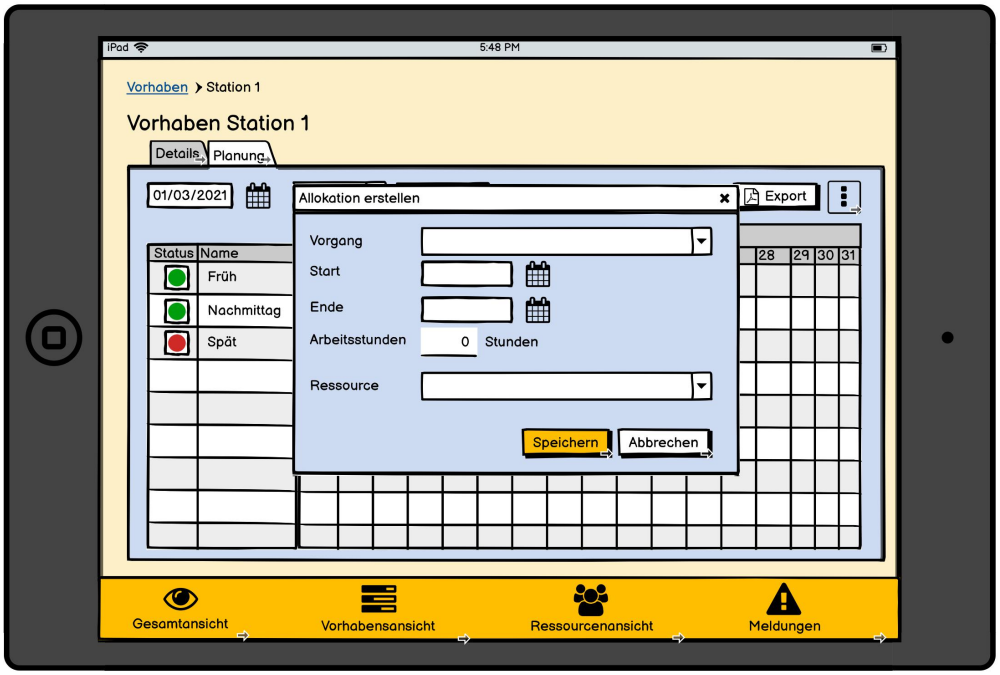
Planer - Vorhabensansicht - Gantt - Message 19 / 28



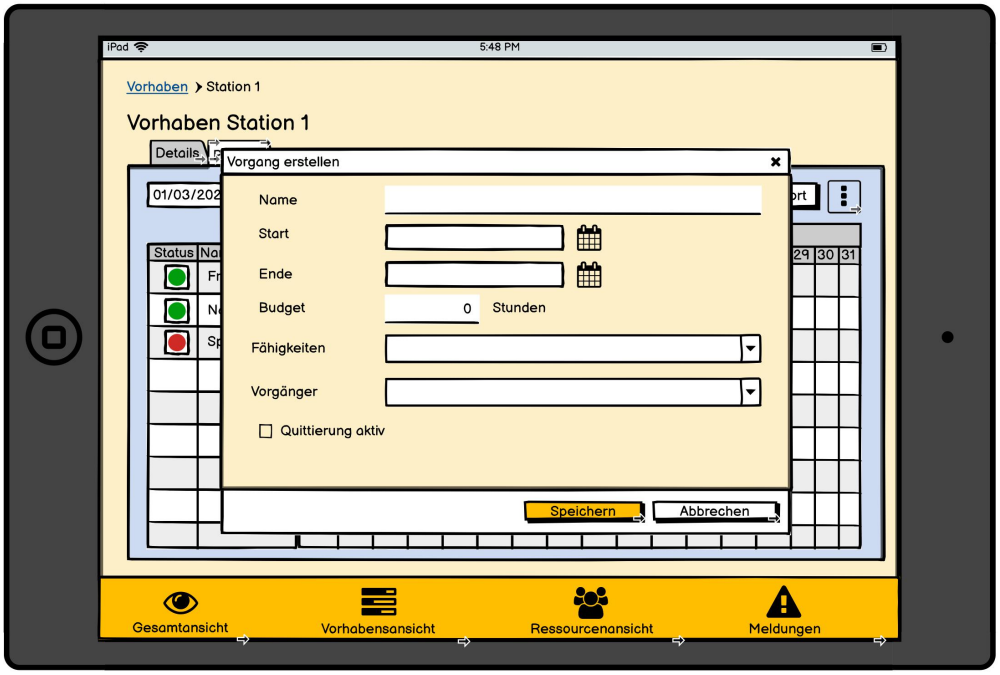
Planer - Vorhabensansicht - Allokation - Edit 20 / 28



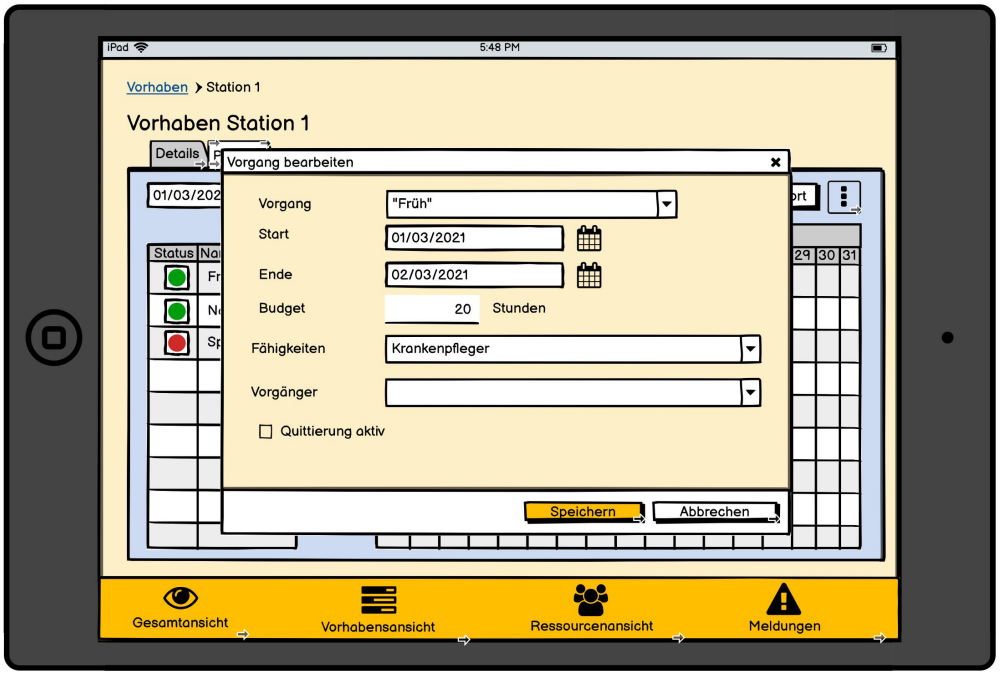
Planer - Vorhabensansicht - Allokation - Create 21 / 28



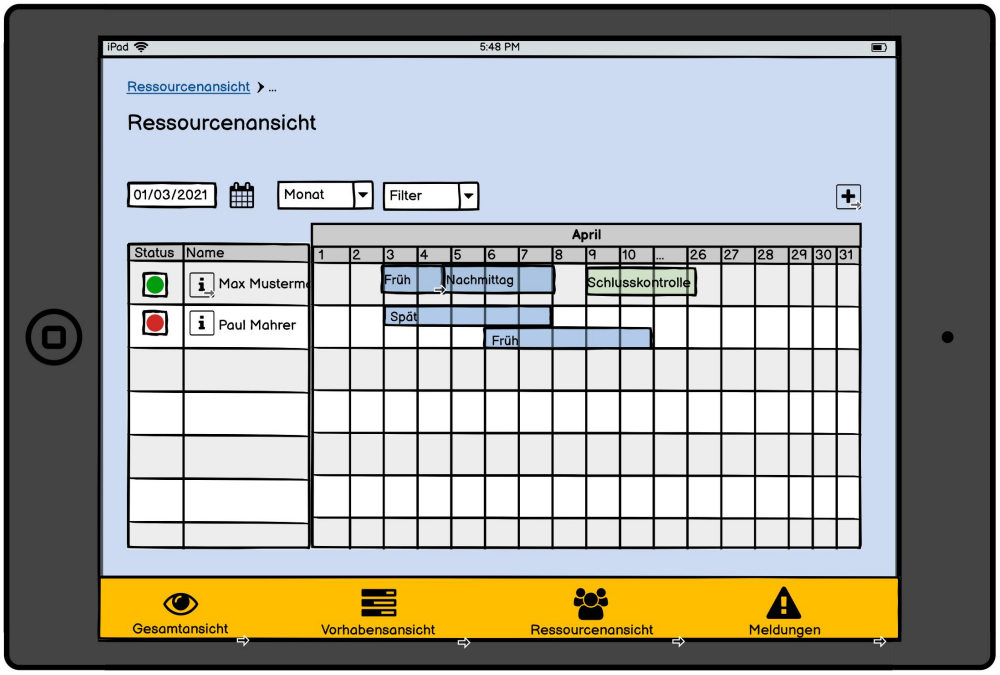
Planer - Vorhabensansicht - Vorgang - Create 22 / 28



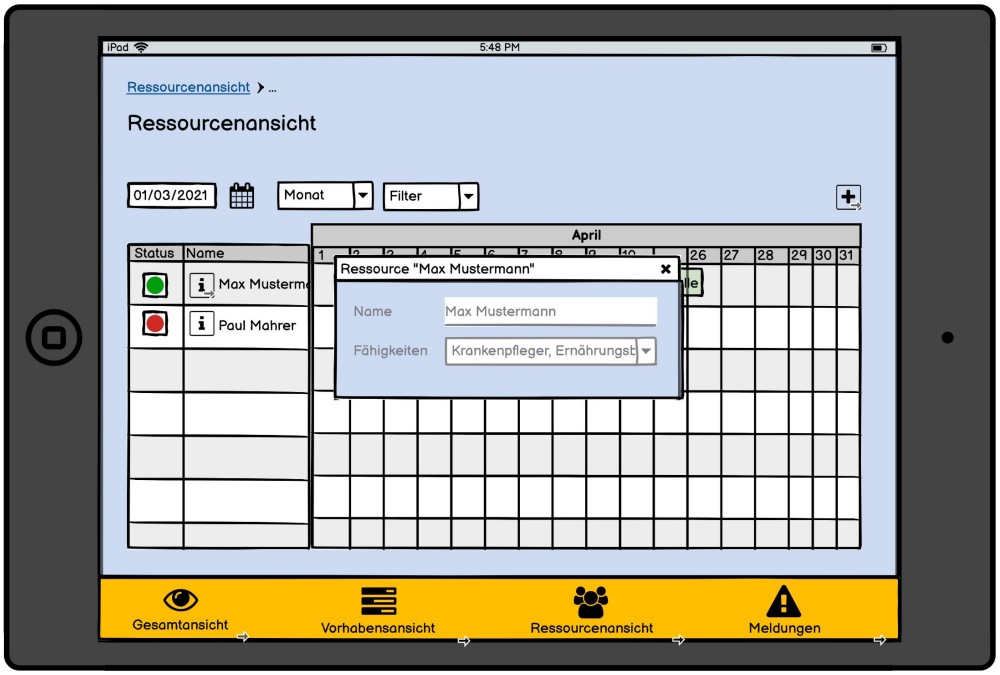
Planer - Vorhabensansicht - Vorgang - Edit 23 / 28



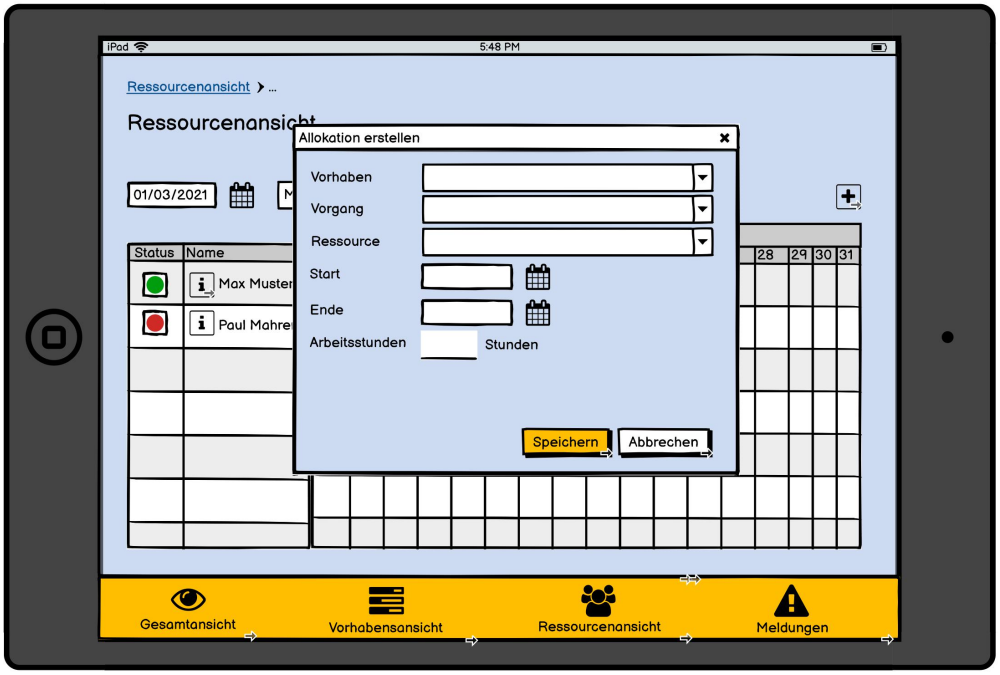
Planer - Ressourcenansicht 24 / 28



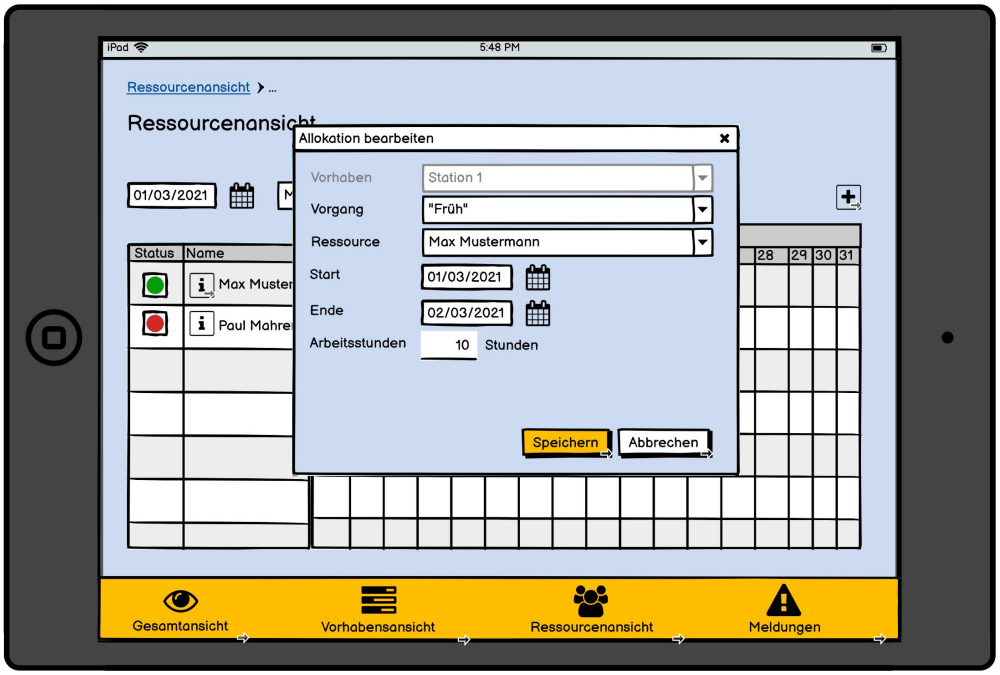
Planer - Ressourcenansicht - Info 25 / 28



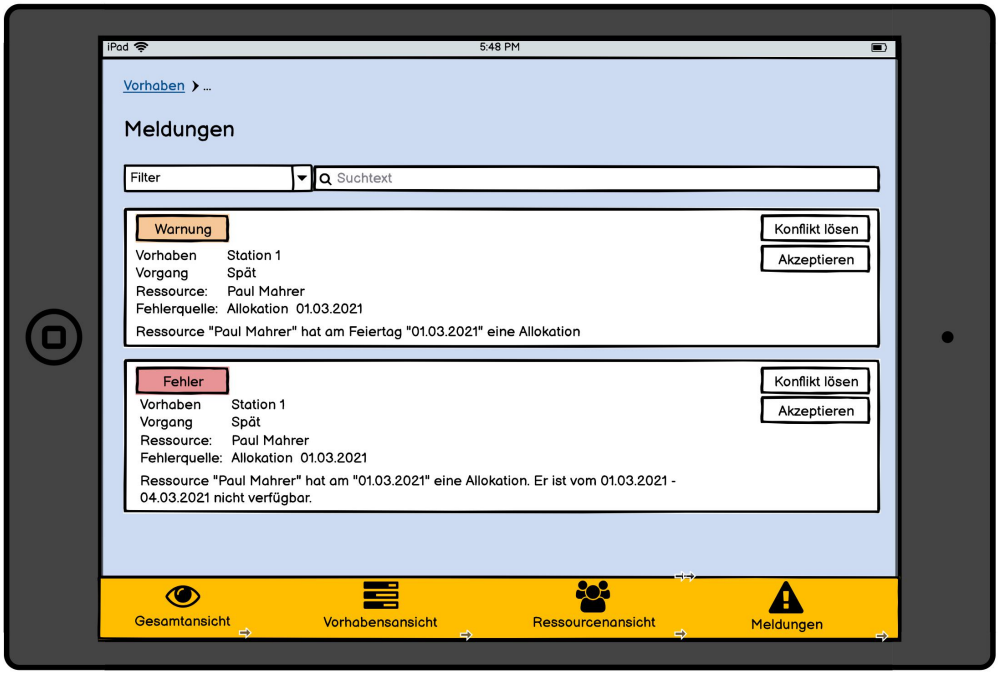
Planer - Ressourcenansicht - Allokation - Create 26 / 28



Planer - Ressourcenansicht - Allokation - Edit 27 / 28



Planer - Messages 28 / 28



Anhang C

Cognitive Walkthrough

C.1 Vorbereitung

Im Rahmen unserer Bachelorarbeit entwickeln wir das Produkt 2PLAN, das es ermöglichen soll, eine Ressourcenplanung durchzuführen. Die Ressourcenplanung umfasst dabei verschiedene Varianten wie eine Projekt-, Schicht- oder Einsatzplanung.

Der erste Teil der Arbeit umfasst die Stammdatenverwaltung. Hier können alle benötigten Daten für die Planung erfasst werden.

Beim zweite Teil der Arbeit geht es um die Ressourcenplanung. Hier wird definiert, wer wann woran arbeitet. Bei der Planung soll grundsätzlich alles möglich sein. Falls es Konflikte in der Planung gibt, sollen diese visuell dargestellt und einfach aufgelöst werden.

Der dritte Teil der Arbeit umfasst den Dienstplan. Beim Dienstplan soll ersichtlich sein, wann eine Ressource für welche Arbeit zugeteilt wurde.

C.2 Szenarios

C.2.1 Szenario 1 - Projekt erstellen

Sie sind Projektleiter in der Baufirma "Mustermann Bau" und haben kürzlich die Zusage für den Hausbau an der Schwalbenstrasse erhalten. Sie wollen nun das Projekt mit allen dazugehörigen Daten im System erfassen.

Beispieldaten:

- Baustelle einrichten
- Aushubarbeit
- Bodenplatte giessen
- Rohbau

C.2.2 Szenario 2 - Projekt planen

Nachdem die Daten alle erfasst sind, geht es nun an die Planung. Weisen Sie den einzelnen Schritten Ressourcen zu, die für die Abarbeitung der Aufgabe zuständig sind. Prüfen Sie, ob bei der Planung Konflikte entstanden sind und beheben Sie diese.

C.2.3 Szenario 3 - Dienstplan

Aus der Sicht des Arbeiters möchten Sie wissen, wie ihr Arbeitsplan für die nächsten zwei Wochen aussieht.

C.3 Durchführung

Wir haben am 06.05.2021 mit der User Experience Designerin, Priska Steiger, einen Review unseres Produktes durchgeführt. Sie hat zusammen mit uns die Szenarien durchgeführt und uns anschliessend Feedback gegeben. Beim Feedback ging es vor allem um visuelle Verbesserungen und um Verbesserungen, die das Arbeiten mit 2PLAN vereinfachen würden.

Nachfolgend ist eine Auflistung aller Verbesserungen zu finden. Gemäss Absprache mit Raphael Ritter müssen nicht alle Verbesserungen im Rahmen der Bachelorarbeit umgesetzt werden. Jedoch soll klar ersichtlich sein, was gemacht wurde und was nicht, damit 2BIT den Rest gegebenenfalls umsetzen kann.

C.3.1 Menu

- ☐ Möglicherweise wäre es sinnvoll, eine Dashboard Seite als Startpunkt zu definieren
- ☐ Wenn man in der Mitte oben auf das 2BIT Icon klickt, würde man erwarten, dass man wieder auf die Landing Page kommt.
- ☐ Das Menü links oben ist sehr versteckt. Ab einer gewissen Bildschrimgrösse würde es mehr Sinn machen, wenn man es immer einblenden würde.
- ☐ Die Benutzereinstellungen können im Menu entfernt werden, da man oben rechts über das Profil drauf kommt.
- ☒ Bei den Seiten neben den Titeln würde es Sinn machen einen Back Button anzuzeigen, um wieder auf die Übersichtsseiten zu gelangen oder Breadcrumb anzuwenden.

C.3.2 Buttons

- ☐ Es soll bei allen Buttons ein Hover geben.

C.3.3 Formulare

- ☒ Bei den Feldern würde es Sinn machen, fixe Längen für kurze und lange Eingaben zu definieren, damit die Struktur etwas harmonischer wirkt.
- ☒ Zwingende Felder mit einem Stern markieren.

C.3.4 Dialoge

- ☒ Die Dialoge vergrössern sich, wenn eine Fehlermeldung angezeigt wird. Hier könnte man eine fixe Höhe setzen, damit es nicht so wirkt, als ob der Dialog herumspringt.
- ☒ Ein Vorgang kann zum Beispiel nur im Datumsbereich des Vorhabens liegen. Entweder könnte man die Kalenderauswahl auf diesen Bereich einschränken oder in der Fehlermeldung den Datumsbereich anzeigen.

C.3.5 Listenansichten

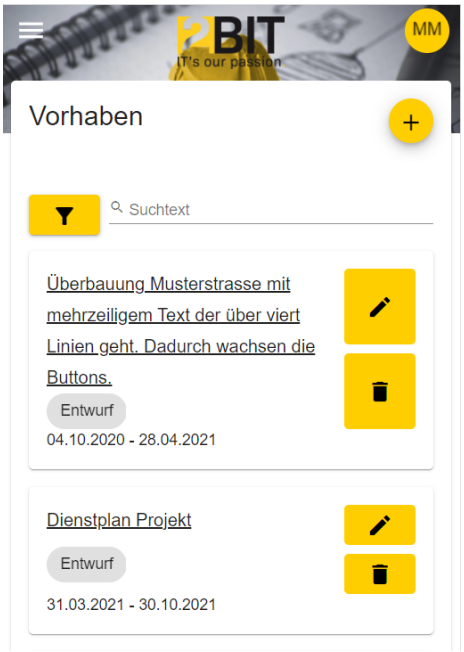


Abbildung C.1: Mobile Listenansicht vom 06.05.2021

- ☒ Wenn es in der Listenansicht einen mehrzeiligen Titel hat, wachsen die Buttons auf der rechten Seite mit. Das sollte nicht sein.
- ☒ Den Edit Button bei den Vorhaben und Vorgängen könnte man entfernen, indem man bei einem Click auf ein Card Element den Bearbeitungsmodus öffnet. Ausserdem hat man aktuell beim unterstrichenen Text das Gefühl, dass man drauf klicken kann und etwas passieren sollte.
- ☒ Der Delete Button sollte besser grau dargestellt werden und nicht so präsent sein.
- ☒ Die Sortierung der Listenelemente ist momentan beliebig. Entweder soll das Startdatum verwendet werden oder der User kann eine Sortierung festlegen.
- ☐ Die Cards sind nicht gut genug ersichtlich, da oben eine Linie fehlt und nur auf den Seiten und unten ein leichter Schatten vorhanden ist. Entweder sollte es einen leicht grauen Hintergrund geben oder es soll runderherum eine feine Linie geben.

C.3.6 Suche

- ☒ Die Suche soll zeitverzögert ausgeführt werden. Momentan flakert der Bildschirm, weil man bei jeder Tasteneingabe die Suche ausführt.
- ☒ Die Suchfelder sollten anders aussehen als normale Input Felder.
- ☒ Wenn es keine Listenelemente gibt, die man durchsuchen kann, sollte die Suche verschwinden.

C.3.7 Planung

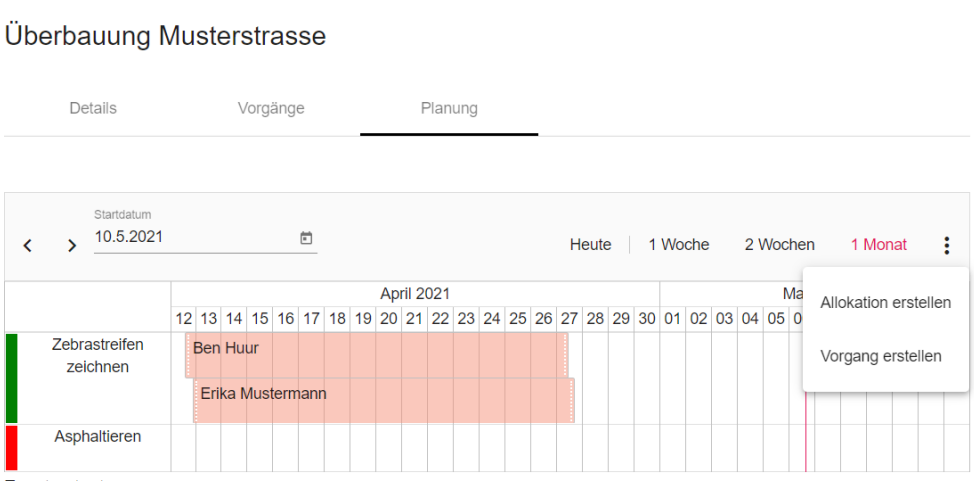


Abbildung C.2: Planungsansicht vom 06.05.2021

- ☒ ”Allokation erstellen” ist zu versteckt. In diesem Dropdown Menu sollten grundsätzlich Aktionen zu finden sein, die nicht so wichtig sind. Besser wäre mit dem gelben Plus Button zu arbeiten, wie auf den anderen Seiten.
- ☒ Es würde die Planung sehr erleichtern, wenn man im Gantt-Diagram auf eine Zelle klicken kann und dann öffnet sich der Dialog, um eine Allokation zu erstellen. Hier wäre es aber gut, wenn die Zelle beim Mouse Over leicht eingefärbt werden würde.
- ☒ Wenn es keine Konflikte gibt, soll der grüne Balken nicht angezeigt werden.
- ☒ Aktuell öffnet sich beim Klick auf den Vorgang (z.B. ”Asphaltieren” im Bild) ein Dialog mit den Vorgangdetails. Beim Klick auf den roten Balken werden die Meldungen für diesen Vorgang in einem Dialog angezeigt. Da der Balken sehr schmal ist, würde es mehr Sinn machen, wenn man beide Informationen in einem Dialog anzeigt. Dann könnte man im Dialog zum Beispiel mit Expansion Panels arbeiten.
- ☒ Das Startdatum soll zwischen den Pfeilen angezeigt werden.
- ☐ Eventuell würde es Sinn machen, nicht nur eine Eingabe für das Startdatum zu haben, sondern eine Eingabe für einen Datumsbereich.
- ☒ Der Button ”Heute” gehört eher zu der Datumsselektion und sollte somit weiter links angezeigt werden. Am Besten wäre es aber, wenn man Heute im Kalender auswählen kann, dann wäre der Button gar nicht nötig.
- ☒ Die aktuelle Selektion wird rot markiert. Möglicherweise wäre eine andere Farbe besser geeignet.
- ☐ Die Buttons (1 Woche, 2 Wochen etc.) sehen aus wie normale Texte. Möglicherweise wäre ein anderes Styling besser.
- ☒ Beim Öffnen der Planungsseite wird als Startdatum immer Today gewählt. Wenn Today zwischen dem Start- / Enddatum des Projekts liegt, ist das in Ordnung. Wenn Today aber ausserhalb liegt, soll das Startdatum des Projekts verwendet werden.

C.3.8 Dienstplan

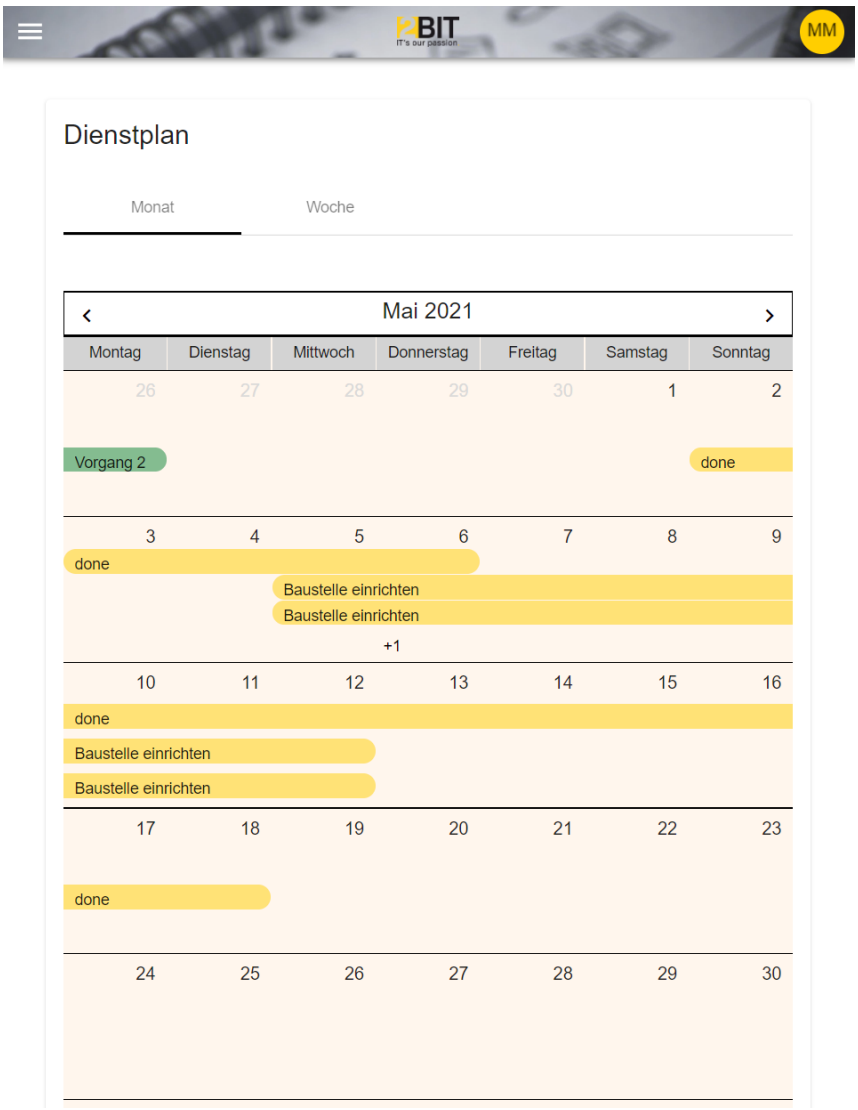


Abbildung C.3: Dienstplan vom 06.05.2021

- ☒ Das Styling sollte mehr an das Styling vom Gantt in der Planung angepasst werden.
- ☒ Der Kalender sollte hellgraue anstatt schwarze Linien verwenden (sowohl vertikal als auch horizontal). Wenn es ausserdem um den ganzen Kalender eine graue Linie gibt, bräuchte es keine Hintergrundfarbe mehr.
- ☒ Der Header (”Mai 2021”) könnte so dargestellt werden wie der Header beim Gantt-Diagramm.
- ☒ Die Tage (Montag, Dienstag, etc.) brauchen keine Hintergrundfarbe.
- ☒ Beim Wechsel von Monat zur Wochenansicht wäre es schön, wenn das Datum übernommen werden würde.
- ☒ Bei der Planung gab es für die verschiedenen Ansichten keine Tabs. Die Ansichten konnten vielmehr über einen Button gewechselt werden. Das würde hier auch Sinn machen.
- ☐ Bei der Wochenansicht wäre es schön, wenn es einen automatischen Scroll geben würde, sodass per Default der Zeitbereich von 07:00 - 17:00 Uhr angezeigt wird.

☒ Bei der Wochenansicht soll der Header immer angezeigt werden, wenn man scrollt.

C.3.9 Meldungen

☒ Die Buttons ”Konflikt lösen” und ”Akzeptieren” sind beide gleichwertig, da aktuell beide Gelb sind. Es muss entschieden werden, was wichtiger ist, und nur dieser Button soll als Primary Button markiert werden. Der andere soll als Secondary definiert werden.

Anhang D

Anwenderdokumentation

Table of Content

Introduction

2PLAN is a Angular Component Library developed for Angular 11, which is used to manage different types of schedules. E.g. project plan, shift schedules and deployment plan. The library uses material design from Angular Material.
It contains a planning component which can visualize the planning on project or resource basis and a calendar component where the resources can look up their schedules.

Getting Started

Installation

1. To use this library you first need to install it with npm: `npm install @2bitgmbh/planning-component`.
2. The library should now be in the `node_modules` folder of your project. Next, we just need to set up the translation before the library is fully functional.
   The translation in the library is implemented with ngx-translate and has its translation files in the following directory `./node_modules/@2Bitgmbh/planning-component/assets/i18n`.
   These files are needed at runtime. To load the translations of the library, the `angular.json` of your project must be modified as follows:

```
 "architect": {
     "build": {
         "options": {
             "assets": [
               {
                 "input": "node_modules/@2bitgmbh/planning-
component/assets/i18n",
                 "glob": "*.json",
                 "output": "assets/i18n/planning-component"
               }
             ]
         }
     }
 }
```

3. After that, you need to use the `LibraryTranslationService` from 2PLAN and call `this.libraryTranslationService.initTranslations(language);` when the language is initialized in your project.

4. The last step is to configure the locale for momentjs. Just add `moment.locale(language);` where your project language is initialized.
5. The code to load the translation from the library could look like this.:

```
this.translateService.use(language).subscribe(() => {
    moment.locale(language);
    this.dateAdapter.setLocale(language);
    this.libraryTranslationService.initTranslations(language);
});
```

Version

Release	Updates
0.2.0	Implemented Library for Angular 11

Build and Test

To build the library you have to be in the client directory of the workspace (./client) and it is required that the Angular CLI is installed. After that, you can enter the following command in the command line: `ng build planning-component`.

The library has no unit tests, but there are some E2E tests with Protractor in the directory `./client/e2e`. The E2E tests can be run locally with the command `npm run e2e-tests-local`. Enter this command on the command line in the client directory of the workspace and the tests will be executed. In order for the tests to work, you also need to start the web application with `ng serve` and run the backend locally with Visual Studio. Make sure to run the web application on port 4200 and the backend on port 5000 or change the corresponding url of the web application and backend in the `./client/e2e/protractor.local.conf.js` file.

Components

lib-gantt-activity

The `lib-gantt-activity` component is used to plan the activities of a project. With this component, allocations for activities can be created and edited. In addition, the activities can be edited directly.

If desired, menu items with additional functions can also be inserted via Content Projection.

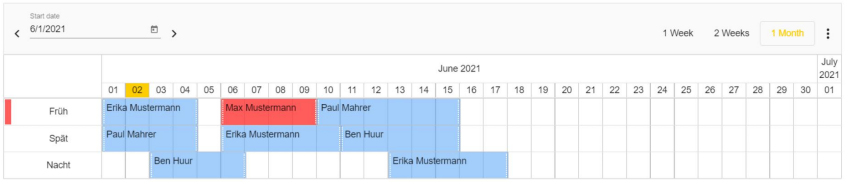
Selector

`lib-gantt-activity`

Example

```
<lib-gantt-activity [activities$]="filteredActivities$"
                    [allocations$]="allocations$"
                    [planningStartDate]="planningStartDate"
                    (editActivity)="editActivity($event)"

 (addAllocationWithPreselectedData)="addAllocationWithPreselectedData($event)"
                    (editAllocation)="editAllocation($event)">
  <button mat-menu-item (click)="addActivity()">{{"ACTIVITY.CREATE" |
translate}}</button>
</lib-gantt-activity>
```



Input

Name	Required	Type	Default	Description
activities$	true	Observable<IActivityDto[]>	`undefined`	Activities which should be displayed as sections.
allocations$	true	Observable<AllocationDto[]>	`undefined`	Allocations which should be displayed as items.
planningStartDate	false	Moment	`moment().startOf("day")`	The start time of the scheduler as a moment object.
lockPast	false	boolean	`false`	If the past is closed and only planning into the future should be allowed.

Output

Name	Type	Description
editActivity	`EventEmitter<string>`	Is called when an activity is clicked. The activity ID is returned.
addAllocationWithPreselectedData	`EventEmitter<{ sectionId: string, startDate: moment_.Moment }>`	Is called when an allocation with preselected data should be added. The activity ID (`sectionId`) and the start date for the allocation are passed.
editAllocation	`EventEmitter<string>`	Is called when an allocation is clicked. The allocation ID is passed.
saveAllocationAfterDragDrop	`EventEmitter<{ id: string, sectionId: string }>`	Is called when an allocation should be saved after drag and drop. Since the allocation can only be moved to another activity, only the allocation ID (`id`) and the new activity id (`sectionId`) are passed.

lib-gantt-resource

The `lib-gantt-resource` component is used for resource scheduling. With this component, allocations for resources can be created and edited. In addition, the resources can be edited directly.

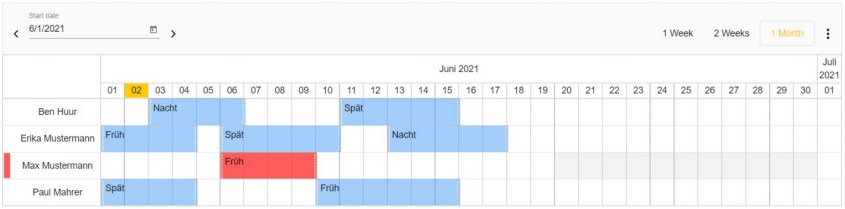
If desired, menu items with additional functions can also be inserted via content projection.

Selector

`lib-gantt-resource`

Example

```
<lib-gantt-resource
  [resources$]="resources$"
  [allocations$]="allocations$"
  [planningStartDate]="planningStartDate"
  (editResource)="editResource($event)"
  (addAllocationWithPreselectedData)="addAllocationWithPreselectedData($event)"
  (editAllocation)="editAllocation($event)">
</lib-gantt-resource>
```



Input

Name	Required	Type	Default	Description
resources$	true	`Observable<IResourceDto[]>`	`undefined`	Resources which should be displayed as sections.
allocations$	true	`Observable<AllocationDto[]>`	`undefined`	Allocations which should be displayed as items.
planningStartDate	false	`Moment`	`moment().startOf("day")`	The start time of the scheduler as a moment object.
lockPast	false	`boolean`	`false`	If the past is closed and only planning into the future should be allowed.

Output

Name	Type	Description
editResource	`EventEmitter<string>`	Is called when a resource is clicked. The resource ID is passed.
addAllocationWithPreselectedData	`EventEmitter<{ sectionId: string, startDate: moment_.Moment }>`	Is called when an allocation with preselected data should be added. The resource ID (`sectionId`) and the start date for the allocation are passed.
editAllocation	`EventEmitter<string>`	Is called when an allocation is clicked. The allocation ID is passed.
saveAllocationAfterDragDrop	`EventEmitter<{ id: string, sectionId: string }>`	Is called when an allocation should be saved after drag and drop. Since the allocation can only be moved to another resource, only the allocation ID (`id`) and the new resource id (`sectionId`) are passed.

lib-gantt

Selector

`lib-gantt [items] [sections]`

Example

```
<lib-gantt [items]="items"
           [sections]="sections"
           [events]="events"
           [periodStart]="planningStartDate">
  <ng-content></ng-content>
</lib-gantt>
```

Input

Name	Required	Type	Default	Description
items	true	`Item[]`	`null`	An array of `Item` to fill up the items of the scheduler.
sections	true	`Section[]`	`null`	An array of `Section` to fill up the sections of the scheduler.
events	false	`Events`	`new Events()`	The events that can be hooked into.
periodStart	false	`Moment`	`moment().startof("day")`	The start time of the scheduler as a moment object.
lockPast	false	`boolean`	`false`	If the past is closed and only planning into the future should be allowed.
locale	false	`string`	`""`	Defines which locale for Moment.js should be loaded. By default, Moment.js uses English (United States) locale strings.
showBusinessDayOnly	false	`boolean`	`false`	Whether only working days are displayed (Mon-Fri).

Models

Section

Sections are displayed on the left side of the scheduler.

Name	Required	Type	Default	Description
id	true	`string`	`null`	A unique identifier for the section (e.q. activity or resource id).
name	true	`string`	`null`	The name to display for the section.
tooltip	false	`string`	`null`	It is used to display tooltip for the section.
absences	false	`IAbsenceDto[] | undefined`	`undefined`	The absences for a section. It is used to display absences in the schedule. This property is useful for resource planning
hasConflicts	false	`boolean`	`false`	Whenever a conflict has arisen in the planning for the section. If it is true a red bar is displayed.

Item

Items are used to display the assignments of a section.

Name	Required	Type	Default	Description
id	true	`number`	`null`	The identifier for the allocation.
name	true	`string`	`null`	The name to display for the item.
start	true	`any`	`null`	A Moment object denoting where this object starts.
end	true	`any`	`null`	A Moment object denoting where this object ends.
sectionID	true	`number`	`null`	The id of the section that this item belongs to.
tooltip	false	`string`	`null`	It is used to display tooltip for the section.
color	false	`string`	`"#ffce00"`	The background color for the allocation. This is useful to visualize the affiliation to a project.
hasConflicts	false	`boolean`	`false`	Whenever a conflict for this allocation exists. If it is true, the background color will be red.

Events

The events are executed on interactions with the items / sections and can be used to react to them.

Name	Parameters	Return type	Description
SectionClickEvent	`section: Section`	`void`	Triggered when a section is clicked.
SectionContextMenuEvent	`section: Section, event: MouseEvent`	`void`	Triggered when a section is righted click.
PeriodChange	`start: moment.Moment, end: moment.Moment`	`void`	Triggered when the period is changed.
AddAllocationWithPreselectedData	`sectionId: string, startDate: moment.Moment`	`void`	Triggered when a cell in the scheduler is clicked.
EditAllocation	`id: string`	`void`	Triggered when an item is clicked.
SaveAllocationAfterDragDrop	`id: string, sectionId: string`	`void`	Triggered when an item is dropped to a new section. `id` is the id of the item and `sectionId` is the id of the target section.

lib-calendar

The `lib-calendar` is used to display a calendar with monthly and weekly view. The calendar is filled with allocations on the corresponding date spans.

Selector

`lib-calendar`

Example

```
<lib-calendar
    [events]="events"
    [monthViewSelected]="true"
    (openAllocation)="openAllocation($event)">
</lib-calendar>
```

Inputs

Name	Required	Type	Default	Description
events	true	`IAllocationDto[]`	`null`	Allocations which are displayed in the calendar.
monthViewSelected	false	`boolean`	`true`	Sets the initial displayed calendar view. If `true` the month view is displayed otherwise the week view is displayed.

Outputs

Name	Type	Description
openAllocation	`EventEmitter<string>`	Is called when an allocation is clicked. The allocation ID is passed.

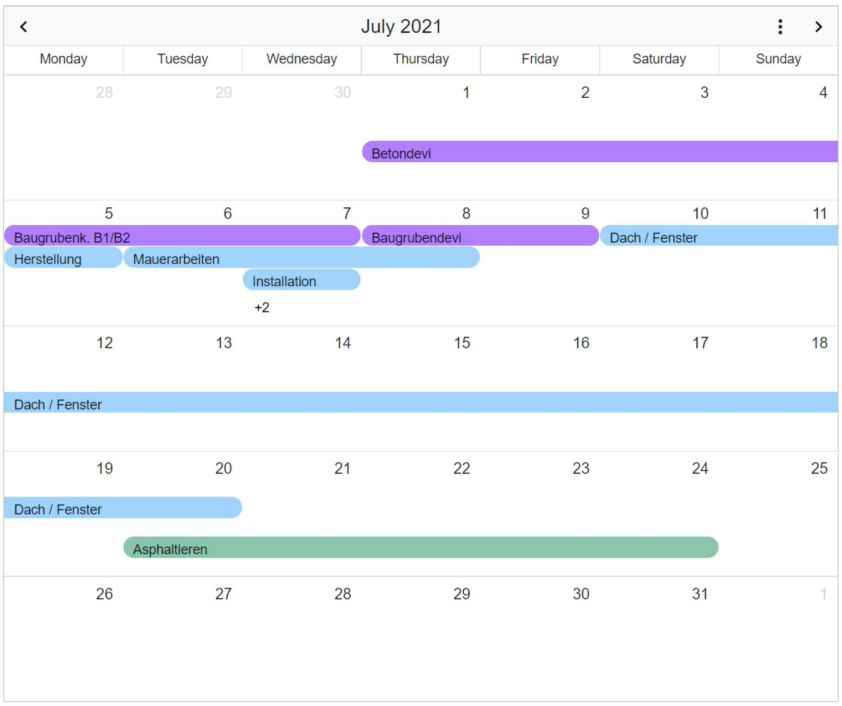
lib-calendar-month

The `lib-calendar-month` displays a calendar monthly view. The calendar is filled with allocations on the corresponding time period. If desired, a navigation in the selection header can be added via content projection.

Example

```
<lib-calendar-month
    *ngIf="monthViewSelected"
    [events]="events"
    [(startDate)]="currentDate"
    (openAllocation)="openAllocationHandler($event)">
  <ng-container>
    <button mat-icon-button [matMenuTriggerFor]="dropMenu">
      <mat-icon>more_vert</mat-icon>
    </button>
    <mat-menu #dropMenu="matMenu">
      <button mat-menu-item
              (click)="changeView('month')"
              [class.selected-view]="monthViewSelected">
        {{ "CALENDAR.MONTH" | translate }}
      </button>
      <button mat-menu-item
              (click)="changeView('week')"
              [class.selected-view]="!monthViewSelected">
        {{ "CALENDAR.WEEK" | translate }}
      </button>
    </mat-menu>
```

```
  </ng-container>
</lib-calendar-month>
```



Inputs

Name	Required	Type	Default	Description
events	true	`IAllocationDto[]`	`null`	Allocations which are displayed in the calendar.
startDate	true	`Date`	`new Date()`	The calendar initially displays the month of the date passed starting with the previous Monday of the first day in that month.

Outputs

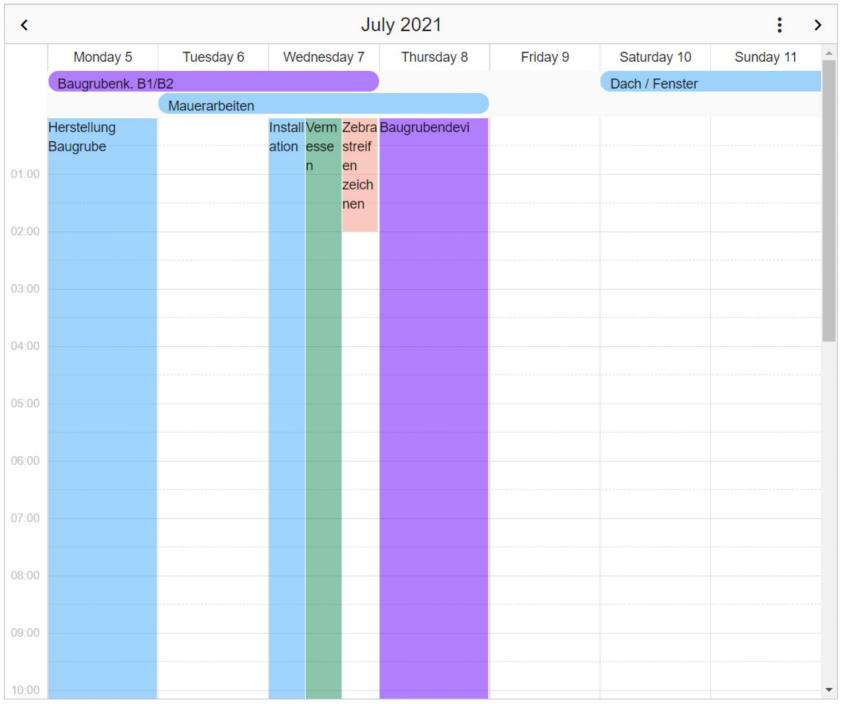
Name	Type	Description
startDateChange	`EventEmitter<Date>`	Is called when the month has been changed. The start date of the new month is passed.
openAllocation	`EventEmitter<string>`	Is called when an allocation is clicked. The allocation ID is passed.

lib-calendar-week

The `lib-calendar-week` displays a calendar weekly view. The calendar is filled with allocations on the corresponding time period. If desired, a navigation in the selection header can be added via content projection.

Example

```
<lib-calendar-week
    [events]="events"
    [(startDate)]="currentDate"
    (openAllocation)="openAllocationHandler($event)">
  <ng-container>
    <button mat-icon-button [matMenuTriggerFor]="dropMenu">
      <mat-icon>more_vert</mat-icon>
    </button>
    <mat-menu #dropMenu="matMenu">
      <button mat-menu-item
              (click)="changeView('month')"
              [class.selected-view]="monthViewSelected">
        {{ "CALENDAR.MONTH" | translate }}
      </button>
      <button mat-menu-item
              (click)="changeView('week')"
              [class.selected-view]="!monthViewSelected">
        {{ "CALENDAR.WEEK" | translate }}
      </button>
    </mat-menu>
  </ng-container>
</lib-calendar-week>
```



Inputs

Name	Required	Type	Default	Description
events	true	`IAllocationDto[]`	`null`	Allocations which are displayed in the calendar.
startDate	true	`Date`	`new Date()`	The calendar initially displays the week of the date passed. Starting at the previous Monday.

Outputs

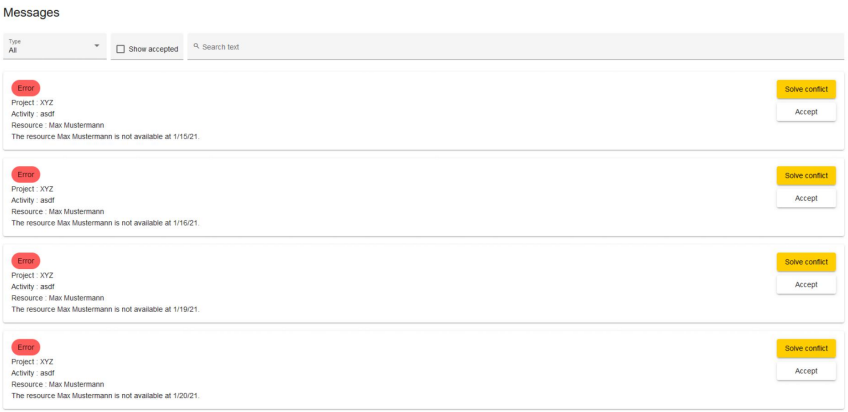
Name	Type	Description
startDateChange	`EventEmitter<Date>`	Is called when the week has been changed. The start date of the new week is passed.
openAllocation	`EventEmitter<string>`	Is called when an allocation is clicked. The allocation ID is passed.

lib-message-overview

The `lib-message-overview` component displays an overview of all messages. They can be filtered by the names of the concerning projects, activities or resources. Furthermore, it provides a filter for the message type and for showing or hiding the accepted messages.

Example

```
<lib-message-overview
    [messages]="data.messages"
    (acceptMessageEvent)="handleAcceptEvent($event)"
    (solveConflictEvent)="handleSolveConflictEvent($event)">
</lib-message-overview>
```



Inputs

Name	Required	Type	Default	Description
messages	true	`IMessageDto[]`	`null`	Messages that should be displayed in the overview.
messageType	false	`number`	`undefined`	Preset for the message type filter.

Outputs

Name	Type	Description
acceptMessageEvent	`EventEmitter<string>`	Returns the message ID of the message that the user clicked to accept.
solveConflictEvent	`EventEmitter<IMessageDto>`	Returns the message that the user wants to solve.

lib-datetime

The `lib-gantt-resource` component can be used if a user should be able to to enter a date and time. If the times are not relevant, the switch "all day" can be activated.

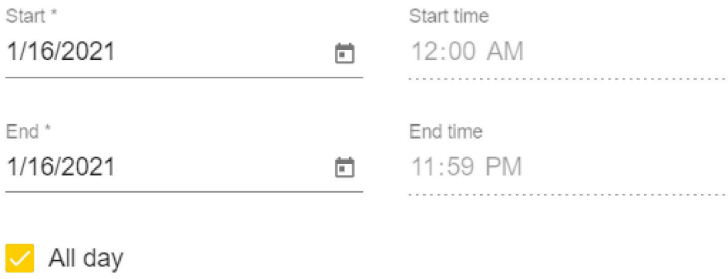
The component ensures that the start must be before the end. In addition, further validations can be defined as input parameters (see `startDateValidator` and `endDateValidator`).

Selector

`lib-datetime [formGroup] [start] [end]`

Example

```
<lib-datetime [formGroup]="activityForm"
              [startParent]="activityProject.startDate"
              [endParent]="activityProject.endDate"
              [start]="activity.startDate"
              [end]="activity.endDate"
              [disabled]="!isEditable"
              [startDateValidator]="isStartDateValid"
              [endDateValidator]="isEndDateValid">
</lib-datetime>
```



Input

Name	Required	Type	Default	Description
formGroup	true	`FormGroup`	`null`	The form group from the outer form element into which the component is integrated. For the input fields, form controls are added to the form group.
start	true	`Date`	`null`	The start date to be displayed in the input field.
end	true	`Date`	`null`	The end date to be displayed in the input field.
startParent	false	`Date | undefined`	`null`	The start date of the parent object. For example, an allocation must be within the date range of an activity. So the start date of the activity is passed here. The passed date is set as min property in the datepicker. Furthermore the date is set as start date if `start` is `undefined`.
endParent	false	`Date | undefined`	`null`	The end date of the parent object. For example, an allocation must be within the date range of an activity. So the end date of the activity is passed here. The passed date is set as max property in the datepicker. Furthermore the date is set as start date if `end` is `undefined`.

Name	Required	Type	Default	Description
startDateValidator	false	`(selectedStart: moment_.Moment) => any`	`null`	`startDateValidator` is executed when the start date changes. If there is no error the function should return `undefined`. Otherwise the function should return an object with a form error (e.g. `{dateRangeProject: true}`). The form error is needed to display an error message below the input field.
endDateValidator	false	`(selectedEnd: moment_.Moment) => any`	`null`	`endDateValidator` is executed when the end date changes. If there is no error the function should return `undefined`. Otherwise the function should return an object with a form error (e.g. `{dateRangeProject: true}`). The form error is needed to display an error message below the input field.
disabled	false	`boolean`	`false`	Whether all fields should be disabled.

NOTE formGroup: The form controls are added to the form group with the following keys:

- "startDate"
- "startTime"
- "endDate"
- "endTime"

StartDate and endDate contains a Moment.js object which combines the entered date and time.

lib-allocation-dialog

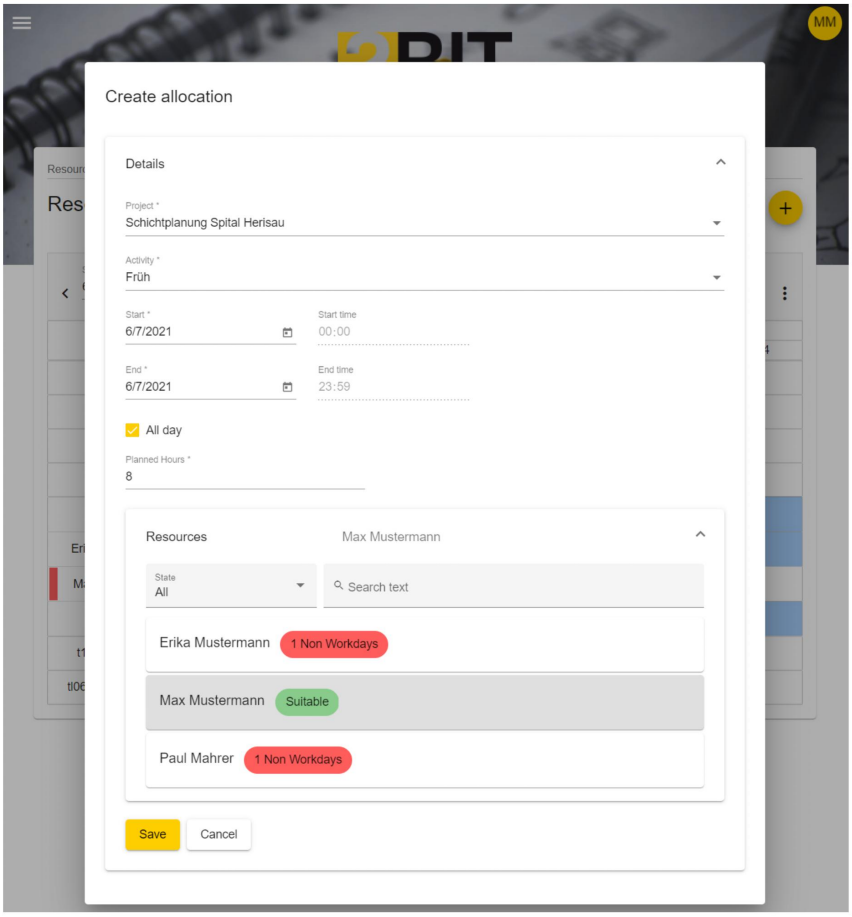
The `lib-allocation-dialog` component can be used to create or edit an allocation via dialog. It should be used in project and resource planning when an allocation is to be created or edited.

Selector

`lib-allocation-dialog`

Example

```
const dialogRef = this.dialog.open(AllocationDialogComponent, {
    width: "80%",
    data: {
        allocation,
        currentProjectId,
        projects$,
        projectActivities$,
        resources$: this.resourcesClient.getResources(),
        loadActivitiesForProjectFunction: this.loadActivitiesForProject,
        lockPast$,
        messages,
    },
});
```



Material Dialog Data

Name	Type	Description
allocation	`IAllocationDto`	The dto of the allocation to be displayed. If a new allocation is to be created, create a new dto.
projects$	`Observable<IProjectDto[]> | undefined`	The projects which can be selected in the dialog. if `undefined` is passed, the project of an allocation cannot be changed.
currentProjectId	`string | undefined`	The project ID to which the allocation activity belongs to. The value should only be specified if `projects$` is not undefined. Otherwise, `undefined` can be passed.
projectActivities$	`Observable<IActivityDto[]> | undefined`	All activities of the currently active project or `undefined` if project is not specified yet.
resources$	`Observable<IResourceDto[]>`	All resources which are available for selection for the event.
loadActivitiesForProjectFunction	`(projectId: string) => Observable<IActivityDto[]>`	Function which is called when the project changes. The function should return all activities of the newly selected project.
lockPast$	`Observable<boolean> | undefined`	Whether the past is closed and only planning into the future is allowed.

Name	Type	Description
messages	`IMessageDto[]`	All messages which belong to the allocation or an empty array if no messages exists.

Public Members

Name	Type	Default	Description
acceptMessageEvent	`EventEmitter<string>`	`new EventEmitter<string>()`	Is called when a message is accepted. The message ID is passed. You have to subscribe to this event and forward the call to the server to save the message.
solveConflictEvent	`EventEmitter<IMessageDto>`	`new EventEmitter<IMessageDto>()`	Is called when a message is to be resolved. You have to subscribe to this event and navigate to the route where the planner can solve the message.

Example

```
const dialogRef = this.dialog.open(AllocationDialogComponent, { ... });
const messageAcceptSubscription =
dialogRef.componentInstance.acceptMessageEvent.subscribe(value => {
    this.handleMessageAcceptEvent(value, dialogRef, allocation.id);
});
const solveConflictEvent =
dialogRef.componentInstance.solveConflictEvent.subscribe(value => {
    this.handleSolveConflictEvent(value);
});
dialogRef.afterClosed().subscribe(result => {
      messageAcceptSubscription.unsubscribe();
    solveConflictEvent.unsubscribe();
});
```

Dialog result

On Save:

```
{ event: "save", data: IAllocationDto }
```

On Cancel:

```
{ event: "cancel" }
```

On Delete

```
{event: "delete", data: IAllocationDto }
```

lib-allocation-calendar-dialog

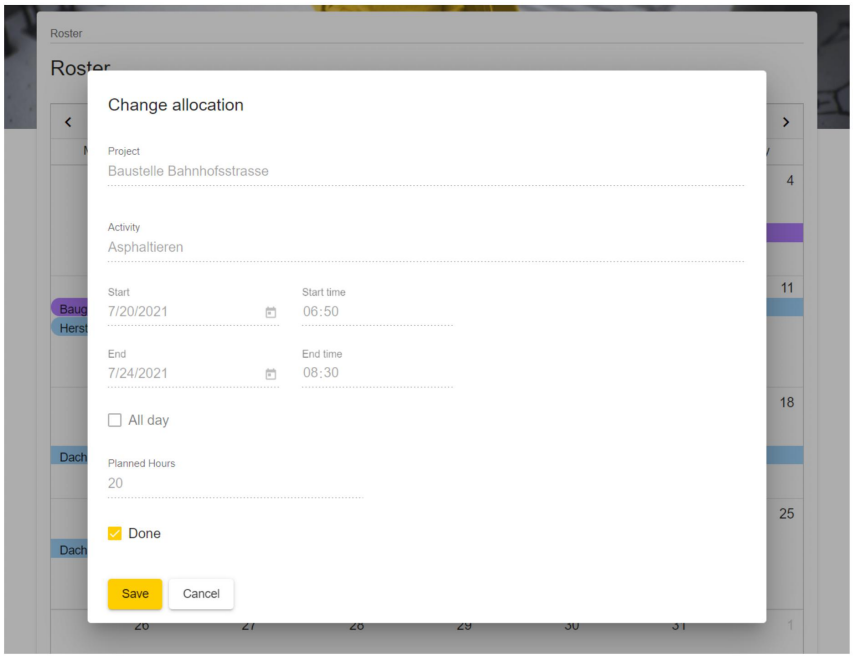
The `lib-allocation-calendar-dialog` component can be used if details of an allocation from the calendar are to be displayed. The allocation is not editable in this dialog. Only the work can be set to done if this has been activated on the activity of the allocation.

Selector

`lib-allocation-calendar-dialog`

Example

```
const dialogRef = this.dialog.open(AllocationCalendarDialogComponent, {
    width: "80%",
    data: {
        allocation: dto,
        isAcknowledgementActive: activityDto.acknowledgement,
    },
});
```



Material Dialog Data

Name	Type	Description
allocation	`IAllocationDto`	The dto of the allocation to be displayed.
isAcknowledgementActive	`boolean`	Whether acknowledgment is active on the activity of the allocation. If true the allocation can be marked as done.

Dialog result

On Save:

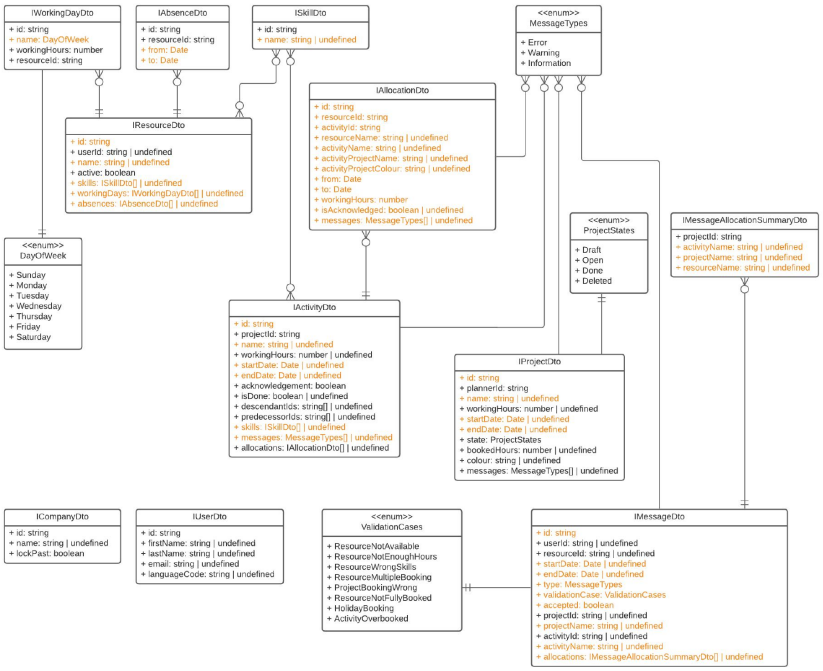
```
{ event: "save", data: IAllocationDto }
```

On Cancel:

```
{ event: "cancel" }
```

View Models

The following class diagram describes the view models and their connections. The properties that are used in the library are highlighted.



IAllocationDto

The `IAllocationDto` is the most important class, which is needed in almost all components. When a resource is assigned to an operation to process it, a corresponding allocation is created. The `IAllocationDto` is needed to display an allocation in the gantt or in the calendar or to display the exact details of an allocation in the dialog.

Name	Type	Description
resourceId	`string`	The id of the resource to which the allocation was assigned.
activityId	`string`	The id of the activity to which the allocation was assigned.
resourceName	`string | undefined`	Is used to display the name of the assigned resource of an allocation in `lib-gantt-activity`.
activityName	`string | undefined`	Is used to display the name of the assigned activity of an allocation.
activityProjectName	`string | undefined`	Is used to display in `lib-allocation-calendar-dialog` the name of the project to which the activity belongs.
activityProjectColour	`string | undefined`	The project color is used to highlight the allocation.
workingHours	`number`	The number of hours a resource has available to complete the work.
isAcknowledged	`boolean | undefined`	Whenever the resource has completed the activity.
messages	`MessageTypes | undefined`	This property is used to find out if there exists messages for an allocation. If so, the allocation is marked red in `lib-gantt`.

IActivityDto

For each project, several activities can be defined. The `IActivityDto` is used to perform the planning and to create `IAllocationDto`.

Name	Type	Description
startDate	`Date | undefined`	When an allocation is created / edited, the start date of the activity is used to check if the allocation has the same or a greater start date. Otherwise the allocation cannot be saved.
endDate	`Date | undefined`	When an allocation is created / edited, the end date of the activity is used to check if the allocation has the same or a smaller end date. Otherwise the allocation cannot be saved.
skills	`ISkillDto | undefined`	Is used to calculate whether the skills of the activity of an allocation match the skills of a resource to determine if the resource is appropriate for the allocation.
messages	`MessageTypes | undefined`	This property is used to find out if there exists messages for an activity. If so, the activity is marked red in `lib-gantt`.

IResourceDto

The `IResourceDto` is used to assign an allocation to a resource.

Name	Type	Description
skills	`ISkillDto | undefined`	Is used to calculate whether the skills of a resource match the skills of the activity of an allocation to determine if the resource is appropriate for the allocation.
workingDays	`IWorkingDayDto[] | undefined`	Is used to calculate whether the days between the start and end dates of an allocation are working days of the resource to determine if the resource is appropriate for the allocation.
absences	`IAbsenceDto[] | undefined`	Is used to calculate if the resource has absences between the days from the start and end dates of an allocation to determine if the resource is appropriate for the allocation.

IMessageDto

Messages are created, when errors occur during planning. These messages are displayed and can be accepted or resolved.

Name	Type	Description
startDate	`Date | undefined`	Defines the start date of a conflict. Only used for the validation cases `ResourceNotAvailable`, `ResourceMultipleBooking`, `HolidayBooking`.
endDate	`Date | undefined`	Defines the end date of a conflict. Only used for the validation cases `ResourceNotAvailable`, `ResourceMultipleBooking`, `HolidayBooking`.
validationCase	`ValidationCase`	Defines why a message exists. Based on the case, an appropriate message is displayed.
projectName	`string | undefined`	Is only used if the message affects only the project. The project name is displayed in the message details.
activityName	`string | undefined`	Is only used if the message affects only the activity of a project. The activity name is displayed in the message details.
allocations	`MessageAllocationSummaryDto[] | undefined`	Defines which allocations are affected by the message. E.g. if there are parallel bookings for a resource on different projects, only one message will be created and the affected allocations will be listed in `allocations`. In `MessageAllocationSummaryDto` the information is stored, which project, which activity and which resource the message concerns.

Anhang E

API Definition

GET baseUrl/user/{id}/company Request, welcher die Company des Users mit der übergebenen Id zurückgibt.

Response:

```
{
    "Id": 1,
    "Name": "Techmania",
    "LockPast": false
}
```

GET baseUrl/company/{id}/user Gibt eine List aller User dieser Company zurück.

Response:

```
{
    "Users": [{
        "Id": 1,
        "LoginId": 5,
        "UserName": "max@muster",
        "Role": 01
    },
    {
        "Id": 2,
        "LoginId": 1,
        "UserName": "admin",
        "Role": 11
    }
}
```

GET baseUrl/user/{id} Gibt der User mit der übergebenen Id zurück.

Response:

```
{
    "User": {
        "Id": 1,
        "LoginId": 5,
        "UserName": "max@muster",
        "Role": 01
    }
}
```

GET baseUrl/company/{id}/vorhaben Gibt eine Liste aller Vorhaben der Company mit der übergebenen Id zurück.

Response:

```
{
    "VorhabenList": [
        {
            "Id": 1,
            "Name": "XYZ",
            "WorkingHours": 40,
            "StartDate": "20-01-2021",
            "EndDate": "30-03-2021",
            "State": 2,
            "BookedHours": 21,
            "MessageIds": [1, 2],
            "VorgaengeIds": [1, 2, 3]
        },
        {
            "Id": 2,
            "Name": "ABC",
            "WorkingHours": 50.5,
            "StartDate": "30-02-2021",
            "EndDate": "25-05-2021",
            "State": 1,
            "BookedHours": 0,
            "MessageIds": null,
            "VorgaengeIds": null
        }
    ]
}
```

GET baseUrl/user/{id}/vorhaben Gibt eine Liste aller Vorhaben zurück, die dem User mit der übergeben Id zugeteilt sind.

Response:

```
{
    "VorhabenList": [
        {
            "Id": 1,
            "Name": "XYZ",
            "WorkingHours": 40,
            "StartDate": "20-01-2021 00:00:00.000 UTC",
            "EndDate": "30-03-2021 00:00:00.000 UTC",
            "State": 2,
            "BookedHours": 21,
            "MessageIds": [1, 2],
            "VorgaengeIds": [1, 2, 3]
        },
        {
            "Id": 2,
            "Name": "ABC",
            "WorkingHours": 50.5,
            "StartDate": "30-02-2021 00:00:00.000 UTC",
            "EndDate": "25-05-2021 00:00:00.000 UTC",
            "State": 1,
            "BookedHours": 0,
            "MessageIds": null,
            "VorgaengeIds": null
        }
    ]
}
```

GET baseUrl/vorhaben/{id} Gibt das Vorhaben mit der übergebenen Id zurück

Response:

```
{
    "Vorhaben": {
        "Id": 1,
        "Name": "XYZ",
        "WorkingHours": 40,
        "StartDate": "20-01-2021 00:00:00.000 UTC",
        "EndDate": "30-03-2021 00:00:00.000 UTC",
        "State": 2,
        "BookedHours": 21,
        "MessageIds": [1, 2],
        "VorgaengeIds": [1, 2, 3]
    }
}
```

PUT baseUrl/vorhaben Übernimmt Änderungen an einem Vorhaben und speichert diese in der DB ab.

Request:

```
{
    "Vorhaben": {
        "Id": 1,
        "Name": "XYZ",
        "StartDate": "20-01-2021 00:00:00.000 UTC",
        "EndDate": "30-03-2021 00:00:00.000 UTC",
        "WorkingHours": 45,
        "State": 2,
        "BookedHours": 21
    }
}
```

POST baseUrl/vorhaben Erstellt ein neues Vorhaben anhand des Json Objekts im Body.

Request:

```
{
    "Vorhaben": {
        "Name": "GHI",
        "StartDate": "25-01-2021 00:00:00.000 UTC",
        "EndDate": "20-03-2021 00:00:00.000 UTC",
        "WorkingHours": 35,
        "State": 1,
        "BookedHours": null
    }
}
```

Response:

```
{
    "Vorhaben": {
        "Id": 3,
        "Name": "GHI",
        "StartDate": "25-01-2021 00:00:00.000 UTC",
        "EndDate": "20-03-2021 00:00:00.000 UTC",
        "WorkingHours": 35,
        "State": 1,
        "BookedHours": null
    }
}
```

DELETE baseUrl/vorhaben/{**id**} Entfernt das Vorhaben mit der übergebenen Id aus der DB.

GET baseUrl/company/{id}/vorgang Gibt alle Vorgänge der Company mit der übergebenen Id zurück.

Response:

```
{
    "Vorgaenge": [
        {
            "Id": 1,
            "Name": "Frueh",
            "WorkingHours": 10.5,
            "StartDate": "20-01-2021 00:00:00.000 UTC",
            "EndDate": "25-01-2021 00:00:00.000 UTC",
            "Acknowledgement": true,
            "isDone": false,
            "DescendantIds": null,
            "PredecessorIds": [2],
            "MessageIds": [1],
            "AllocationIds": [1, 2]
        },
        {
            "Id": 2,
            "Name": "Spaet",
            "WorkingHours": 10.5,
            "StartDate": "26-01-2021 00:00:00.000 UTC",
            "EndDate": "30-01-2021 00:00:00.000 UTC",
            "Acknowledgement": false,
            "isDone": false,
            "DescendantIds": [1],
            "PredecessorIds": null,
            "MessageIds": null,
            "AllocationIds": null
        }
    ]
}
```

GET baseUrl/vorgang/{id} Gibt den Vorgang mit der übergebenen Id zurück.

Response:

```
{
    "Vorgang": {
        "Id": 1,
        "Name": "Frueh",
        "WorkingHours": 10.5,
        "StartDate": "20-01-2021 00:00:00.000 UTC",
        "EndDate": "25-01-2021 00:00:00.000 UTC",
        "Acknowledgement": true,
        "isDone": false,
        "DescendantIds": null,
        "PredecessorIds": [2],
        "MessageIds": [1],
        "AllocationIds": [1, 2]
    }
}
```

PUT baseUrl/vorgang Übernimmt Änderungen an einem Vorgang und speichert diese in der DB ab.

Request:

```
{
    "Vorgang": {
        "Id": 1,
```

```
        "Name": "Frueh",
        "WorkingHours": 12.5,
        "StartDate": "22-01-2021 00:00:00.000 UTC",
        "EndDate": "25-01-2021 00:00:00.000 UTC",
        "Acknowledgement": true,
        "isDone": false,
        "DescendantIds": null,
        "PredecessorIds": [2]
    }
}
```

POST baseUrl/vorgang Erstellt einen neuen Vorgang anhand des Json Objekts im Body.

Request:

```
{
    "Vorgang": {
        "Name": "Nacht",
        "WorkingHours": 8.5,
        "StartDate": "20-01-2021 00:00:00.000 UTC",
        "EndDate": "25-01-2021 00:00:00.000 UTC",
        "Acknowledgement": true,
        "isDone": false,
        "DescendantIds": null,
        "PredecessorIds": null
    }
}
```

Response:

```
{
    "Vorgang": {
        "Id": 3,
        "Name": "Nacht",
        "WorkingHours": 8.5,
        "StartDate": "20-01-2021 00:00:00.000 UTC",
        "EndDate": "25-01-2021 00:00:00.000 UTC",
        "Acknowledgement": true,
        "isDone": false,
        "DescendantIds": null,
        "PredecessorIds": null
    }
}
```

DELETE baseUrl/vorgang/{**id**} Entfernt den Vorgang mit der übergebenen Id aus der DB.

GET baseUrl/company/{**id**}**/message** Gibt alle Messages der Company mit der übergebenen Id zurück.

Response:

```
{
    "Messages": [
        {
            "Id": 1,
            "Type": 1,
            "accepted": false,
            "validationCase": 2,
            "VorhabenIds": [1],
            "VorgaengeIds": [2],
            "AllocationIds": null,
            "RessourceId": null
        },
```

```
        {
            "Id": 2,
            "Type": 2,
            "accepted": false,
            "validationCase": 3,
            "VorhabenIds": [2],
            "VorgaengeIds": [2],
            "AllocationIds": null,
            "RessourceId": null
        },
        {
            "Id": 3,
            "Type": 3,
            "accepted": false,
            "validationCase": 4,
            "VorhabenIds": [3, 4],
            "VorgaengeIds": [2, 8],
            "AllocationIds": [2, 5],
            "RessourceId": 3
        }
    ]
}
```

GET baseUrl/message/{**id**} Gibt die Message mit der übergebenen Id zurück.

Response:

```
{
    "Message": {
        "Id": 2,
        "Type": 2,
        "accepted": false,
        "validationCase": 1,
        "VorhabenIds": [2],
        "VorgaengeIds": [2],
        "AllocationIds": null,
        "RessourceId": null
    }
}
```

PUT baseUrl/message Übernimmt Änderungen an einer Message und speichert diese in der DB ab.

Request:

```
{
    "Message": {
        "Id": 2,
        "accepted": true
    }
}
```

POST baseUrl/message Erstellt eine neue Message anhand des Json Objekts im Body.

Request:

```
{
    "Message": {
        "Type": 2,
        "accepted": false,
        "validationCase": 1,
        "VorhabenId": 1,
        "VorgangId": 1,
```

```
        "AllocationId": null,
        "RessourceId": null
    }
}
```

Response:

```
{
    "Message": {
        "Id": 2,
        "Type": 2,
        "accepted": false,
        "validationCase": 1,
        "VorhabenId": 1,
        "VorgangId": 1,
        "AllocationId": null,
        "RessourceId": null
    }
}
```

DELETE baseUrl/message/{**id**} Entfernt die Message mit der übergebenen Id aus der DB.

GET baseUrl/skill Gibt eine Liste aller Skills zurück.

Response:

```
{
    "Skills": [{
        "Id": 1,
        "Name": "Security"
    },
    {
        "Id": 2,
        "Name": "DevOps"
    }]
}
```

GET baseUrl/company/{**id**}**/ressource** Gibt eine Liste aller Ressourcen der Company mit der übergebenen Id zurück.

Response:

```
{
    "Ressources": [{
        "Id": 1,
        "Name": "Max Muster",
        "Active": true,
        "MessageIds": [4],
        "WorkDays": [{
            "Id": 1,
            "Name": 2,
            "WorkingHours": 8
        }],
        "Absences": [{
            "Id": 1,
            "From": "20-06-2021 00:00:00.000 UTC",
            "To": "30-06-2021 00:00:00.000 UTC"
        }],
        "SkillIds": [1, 2]
    },
    {
        "Id": 2,
        "Name": "Ben Huur",
```

```
        "Active": true,
        "Messages": null,
        "WorkDays": [{
            "Id": 2,
            "Name": 3,
            "WorkingHours": 4.5
        }],
        "Absences": null,
        "SkillIds": [2]
    }]
}
```

GET baseUrl/ressource/{id} Gibt die Ressource der übergebenen Id zurück.

Response:

```
{
    "Ressource": {
        "Id": 1,
        "Name": "Max Muster",
        "Active": true,
        "MessageIds": [3],
        "WorkDays": [{
            "Id": 1,
            "Name": 2,
            "WorkingHours": 8
        }],
        "Absences": [{
            "Id": 1,
            "From": "20-06-2021 00:00:00.000 UTC",
            "To": "30-06-2021 00:00:00.000 UTC"
        }],
        "SkillIds": [1, 2]
    }
}
```

GET baseUrl/user/{id}/ressource Gibt die zugehörige Ressource des Users mit der übergebenen Id zurück.

GET baseUrl/company/{id}/allocation Gibt eine Liste aller Allokationen der Company mit der übergebenen Id zurück.

Response:

```
{
    "Allocations": [{
        "Id": 1,
        "RessourceId": 1,
        "VorgangId": 1,
        "From": "21-01-2021 00:00:00.000 UTC",
        "To": "25-01-2021 00:00:00.000 UTC",
        "WorkingHours": 8,
        "IsAcknowledged": false,
        "MessageIds": [5]
    },
    {
        "Id": 2,
        "RessourceId": 1,
        "VorgangId": 2,
        "From": "21-01-2021 00:00:00.000 UTC",
        "To": "25-01-2021 00:00:00.000 UTC",
        "WorkingHours": 8,
        "IsAcknowledged": false,
```

```
        "MessageIds": [6,7]
    }]
}
```

GET baseUrl/allocation/{id} Übernimmt Änderungen an einer Allokation und speichert diese in der DB ab.

Response:

```
{
    "Allocation": {
        "Id": 1,
        "RessourceId": 1,
        "VorgangId": 1,
        "From": "21-01-2021 00:00:00.000 UTC",
        "To": "25-01-2021 00:00:00.000 UTC",
        "WorkingHours": 8,
        "IsAcknowledged": false,
        "MessageIds": [5]
    }
}
```

PUT baseUrl/allocation Übernimmt Änderungen an einer Allokation und speichert diese in der DB ab.

Request:

```
{
    "Allocation": {
        "Id": 1,
        "RessourceId": 1,
        "VorgangId": 1,
        "From": "21-01-2021 00:00:00.000 UTC",
        "To": "25-01-2021 00:00:00.000 UTC",
        "WorkingHours": 8,
        "IsAcknowledged": false,
        "MessageIds": [5]
    }
}
```

POST baseUrl/allocation Erstellt eine neue Allokation anhand des Json Objekts im Body.

Request:

```
{
    "Allocation": {
        "RessourceId": 1,
        "VorgangId": 1,
        "From": "21-01-2021 00:00:00.000 UTC",
        "To": "25-01-2021 00:00:00.000 UTC",
        "WorkingHours": 8,
        "IsAcknowledged": false,
        "MessageIds": null
    }
}
```

Response:

```
{
    "Allocation": {
        "Id": 2,
        "RessourceId": 1,
        "VorgangId": 1,
```

```
        "From": "21-01-2021 00:00:00.000 UTC",
        "To": "25-01-2021 00:00:00.000 UTC",
        "WorkingHours": 8,
        "IsAcknowledged": false,
        "MessageIds": null
    }
}
```

DELETE baseUrl/allocation/{**id**} Entfernt die Allokation mit der übergebenen Id aus der DB.

Literaturverzeichnis

[1] *.NET*. URL: https://dotnet.microsoft.com/.

[2] *Angular*. URL: https://angular.io/.

[3] *Angular Calendar*. URL: https://angular-calendar.com/#/kitchen-sink.

[4] *Angular CLI*. URL: https://angular.io/cli.

[5] *Angular Internationalization (i18n)*. URL: https://angular.io/guide/i18n.

[6] *Angular Material*. URL: https://material.angular.io/.

[7] *Angular Material Progress Spinner*. URL: https://material.angular.io/components/progress-spinner/overview. (Zugegriffen am 01.06.2021).

[8] *Angular Material Scrolling*. URL: https://material.angular.io/cdk/scrolling.

[9] *Azure DevOps 2BIT*. URL: https://2bit.visualstudio.com/2PLAN.

[10] *Code Zeilen Statistik*. URL: https://medium.com/modern-stack/how-much-computer-code-has-been-written-c8c03100f459.

[11] *Data Management mit NgRx*. URL: https://ngrx.io/.

[12] *ESLint*. URL: https://eslint.org.

[13] *FullCalendar*. URL: https://fullcalendar.io/.

[14] *Gantt chart component for AngularJS*. URL: https://www.angular-gantt.com/.

[15] *gantt-schedule-timeline-calendar*. URL: https://gantt-schedule-timeline-calendar.neuronet.io/.

[16] *GitLab*. URL: https://about.gitlab.com/.

[17] *Guidelines Material Design*. URL: https://material.io/design/guidelines-overview.

[18] *Infinite Scrolling vs Virtual Scrolling*. URL: https://javascript.plainenglish.io/infinite-scrolling-vs-virtual-scrolling-in-angular-57f6e60f1285. (Zugegriffen am 01.06.2021).

[19] *Ionic2 Calendar*. URL: https://github.com/twinssbc/Ionic2-Calendar.

[20] *Lokale Angular Library*. URL: https://angular.io/guide/creating-libraries.

[21] *Nuget Package Manager*. URL: https://docs.npmjs.com/about-npm.

[22] *Overleaf*. URL: https://de.overleaf.com/learn.

[23] *Prettier*. URL: https://prettier.io/.

[24] *Protractor*. URL: https://www.protractortest.org/#/.

[25] 2BIT GmbH Raphael Ritter. *Dokument "Anforderungen 2BIT.pdf"*.

[26] *Simple Angular Time Scheduler (ngx-time-scheduler)*. URL: https://github.com/abhishekjain12/ngx-time-scheduler/tree/v1.5.1.

[27] *Sloc*. URL: https://www.npmjs.com/package/sloc.

[28] *Telerik Calendar*. URL: https://www.telerik.com/kendo-angular-ui/calendar.

[29] *Third Party Internationalization Library*. URL: https://github.com/ngx-translate/core.

[30] *TMetric - Time Tracking*. URL: https://tmetric.com/.

[31] *TypeScript*. URL: https://www.typescriptlang.org/.

[32] *Unit Tests nach Martin Fowler*. URL: https://martinfowler.com/bliki/UnitTest.html.

[33] *Zeiterfassung*. URL: https://gitlab.ost.ch/severin.amacher/2PLAN.