Mitarbeiter-Informationsapp für die Pronto AG

Bachelorarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Frühlingssemester 2021

Autoren: Carlo Kirchmeier, Yannick Vogt

Betreuer: Mirko Stocker

Experte: Leo Büttiker

Projektpartner: Pronto AG

Abstract

Diese Arbeit befasst sich mit dem Aufbau einer Mitarbeiter-Informationsapp, welche die Pronto AG zur Verteilung von Informationen an ihre Mitarbeitenden einsetzen möchte [1]. Um der Pronto-AG diese Funktionalität zu ermöglichen, wurde im Verlauf dieser Arbeit eine cross-platform Applikation, sowie ein Server-Backend entwickelt, welche die gewünschten Funktionen anbieten. Ein wichtiger Bestandteil der Umsetzung war, dass die Nutzer der App zeitnah und für sie gut ersichtlich informiert werden. Daher war die Implementation von Push-Benachrichtigungen von Beginn weg ein integraler Bestandteil dieser Arbeit.

Ausschlaggebend für den Projektauftrag war, dass die Pronto AG den bisherigen Prozess der Informationsverteilung über WhatsApp und Aushänge nicht mehr als zeitgemäss erachtete. Darüber hinaus, konnte mit dieser Methode nicht gut koordiniert werden, dass alle Mitarbeitenden die selben Informationen zur Verfügung hatten. Die Pronto AG erhofft sich, dass durch die neue Applikation eine bessere und einheitlichere Kommunikation mit den Mitarbeitenden möglich wird.

Als Ergebnis dieser Arbeit, entstand ein Server-Backend, geschrieben in C#, sowie eine Frontend-Applikation namens „Pronto MIA“, welche auf den Plattformen Web, Android und iOS läuft. Dies dank der Verwendung des Flutter-Frameworks, welches es ermöglicht, native Apps für all diese Plattformen aus einer Codebase zu erstellen. Zusammen sind das Backend und das Frontend in der Lage, Mitarbeitende über neue Einsatzpläne zu informieren und der Administration zu erlauben, Mitarbeitende, sowie Einsatzpläne zu verwalten. Dank der integrierten Push-Benachrichtigungen werden Mitarbeitende umgehend über neue oder geänderte Einsatzpläne informiert und können diese innerhalb der Applikation direkt abrufen.

Danksagung

Wir möchten unserem Betreuer Mirko Stocker unseren Dank dafür aussprechen, dass er uns, während unserer Arbeit, stets mit wertvollen Tipps zur Seite gestanden hat. Auch danken wir ihm für die schnelle Antworten auf unsere Fragen und seine unkomplizierte Herangehensweise an das Projekt.

Des Weiteren möchten wir Ramon und Heidi Herzog, von der Pronto AG, für die gute Zusammenarbeit danken. Wir schätzten ihre unkomplizierte Art, sowie die flexible Terminfindung immer sehr. Auch sind wir um das erhaltene konstruktive Feedback während den Sitzungen dankbar, dieses hat viel zur Verbesserung der Applikation beigetragen.

Inhaltsverzeichnis

1 Management Summary

In diesem Kapitel wird mithilfe eines Management Summary das Projekt kurz erläutert und ein Überblick darüber geschaffen.

1.1 Ausgangslage

Die Pronto AG ist eine Reinigungsfirma mit über 360 Mitarbeitenden. Diese begeben sich regelmässig zu Kunden um diverse Reinigungs- und Unterhaltsarbeiten durchzuführen. Um diese Einsätze und Aufträge zu koordinieren, nutzt die Pronto AG Einsatzpläne, welche an Aushängen in der Firma für die Mitarbeitenden einsehbar sind. Diese Einsatzpläne bilden die Grundlage für die Verteilung der Arbeitskräfte am darauffolgenden Tag.

Diese Bachelorarbeit befasst sich mit der Digitalisierung der zuvor genannten Einsatzpläne. Hierbei sollen die Mitarbeitenden über ihr Smartphone die Einsatzpläne abrufen können. Die hierzu entwickelte App soll ausserdem noch andere Funktionalitäten, wie beispielsweise das Abwickeln der Ferienanträge enthalten.

1.2 Vorgehen

Im Vorfeld der Implementation wurde mit der Pronto AG eine Analyse der benötigten und gewünschten Funktionalitäten vorgenommen. Daraus wurde anschliessend ein Releaseplan erstellt.

Darauffolgend wurde ein technischer Prototyp umgesetzt, um die technische Machbarkeit der kritischen Funktionen zu überprüfen und zu beweisen.

Es wurde nach der Umsetzung des Prototyps, in enger Zusammenarbeit mit dem Kunden, die gewünschte Applikation entwickelt. Hierbei diente der Releaseplan und das Feedback des Kunden als Leitlinie bei der Entwicklung.

1.3 Ergebnisse

Das Ergebnis dieser Arbeit nennt sich „Pronto MIA“. Der Name steht für „Pronto Mitarbeiter Informations App“. Die Applikation besteht aus einem C# Backend mit GraphQL-API und einem, auf Flutter basierenden, Frontend. Mithilfe der Applikation ist die Pronto AG in der Lage, Einsatzpläne zu erstellen und diese an ihre Mitarbeitenden zu verteilen. Hierbei werden Push-Benachrichtigungen eingesetzt und die Einsatzpläne werden Abteilungsspezifisch verteilt.

„Pronto MIA“ läuft auf den Plattformen Android, iOS und im Web. Die Applikation kann vom Play-Store heruntergeladen oder im Web bedient werden.

1.4 Ausblick

„Pronto MIA“ bietet bisher nur einen Bruchteil der Funktionalität an, welche sich die Pronto AG für die Applikation vorstellen kann. Obschon die wichtigsten Funktionalitäten implementiert wurden, gibt es dennoch viel Potenzial in der Applikation. Das Ausrollen der Applikation in den Apple App-Store konnte im Rahmen dieser Arbeit aufgrund von Limitationen, welche Apple definiert hat, nicht umgesetzt werden. Aufgrund dessen, dass viele Mitarbeitende der Pronto AG Apple-Geräte besitzen, wäre es empfehlenswert, das Ausrollen demnächst in Angriff zu nehmen. Es ist allerdings schon bekannt, dass es eine Folgearbeit für dieses Projekt geben wird, welche sich mit eben diesen Thematiken beschäftigen wird. Da das Ausrollen auf Android und im Web gut geklappt hat, sind hier keine weiteren Massnahmen notwendig.

Teil I

Projektplan

2 Übersicht

Im Teil „Projektplan“ wird die Aufgabenstellung erläutert, der Zeitplan definiert und das Projektvorgehen geplant. Ausserdem werden Projektprozesse, sowie das Risiko und Qualitätsmanagement spezifiziert. Zusammengefasst geht es in diesem Teil um organisatorische Aspekte, welche vor dem technischen Teil des Projektes definiert werden müssen.

2.1 Aufgabenstellung

Die Pronto AG ist ein Komplettanbieter für Reinigungen aller Art in der Ost- und Innerschweiz. Um die Kommunikation mit den rund 360 Mitarbeitenden zu optimieren, soll eine cross-platform Mobile App entwickelt werden.

Folgende Features sind denkbar und sollen in einer agilen Vorgehensweise (der genaue Funktionsumfang wird zusammen mit dem Industriepartner festgelegt, bzw. ergibt sich aus dem Projektfortschritt) umgesetzt werden:

- Einsatzpläne den Mitarbeitenden per Push-Benachrichtigung kommunizieren
- Frei- und Ferienanträge einreichen
- Bereich für Schulungsvideos
- Bereich für News / Kalender / Informationen

Die Mitarbeitenden sind in verschiedenen Abteilungen tätig, entsprechend wird auch eine entsprechende Benutzerverwaltung benötigt.

Diese Aufgabenstellung wurde aus der Original-Aufgabenstellung entnommen.

2.2 Erwartete Resultate

- Software in Form eines cross-platform Mobile Apps, welches die in der vorgegeben Zeit möglichen Features, in einem dem Kunden zufriedenstellenden Umfang, beinhaltet.
- Dokumentation der Arbeit (inklusive Vorgehen und kritische Bewertung der getroffenen Entscheide).

2.3 Einschränkungen

Das Modul „Bachelorarbeit“ ist zeitlich begrenzt. Das Team hat gemeinsam 720h Zeit, die Projektarbeit in 17 Wochen durchzuführen. Diese zeitliche Begrenzung setzt sich folgendermassen zusammen:

- 720h = 2 Teammitglieder * 12 ECTS Punkte * 30h pro ECTS
- 17 Wochen = 15 Semesterwochen + 2 Zusätzliche Wochen exklusiv die Bachelorarbeit

Die Bachelorarbeit beginnt offiziell am 22.02.2021 und endet mit der Abgabe am 18.06.2021 um 12:00 Uhr.

2.4 Persönliche Ziele

Neben dem Endprodukt umfasst das Projekt folgende persönliche Ziele des Teams:

- Erfolgreiche Umsetzung eines vorgegebenen Projekts in der zur Verfügung gestellten Zeit.
- Erstellen eines Endproduktes, welches einen realen Nutzen darstellt und ein Bedürfnis befriedigt.
- Erfahrungen sammeln für weitere schulische, private und berufliche Projekte.
- Kennenlernen der Entwicklung von cross-platform Mobile Apps.
- Kompetenzen im Bereich Kundeninteraktion erweitern.

3 Zeitplan

In diesem Kapitel wird näher auf den zeitlichen Aspekt und die geplanten Schritte eingegangen. Ebenfalls werden Meilensteine definiert und erläutert. Zusätzlich wird auf die verschiedenen Phasen nach Scrum + eingegangen und diese näher erläutert.

3.1 Übersicht

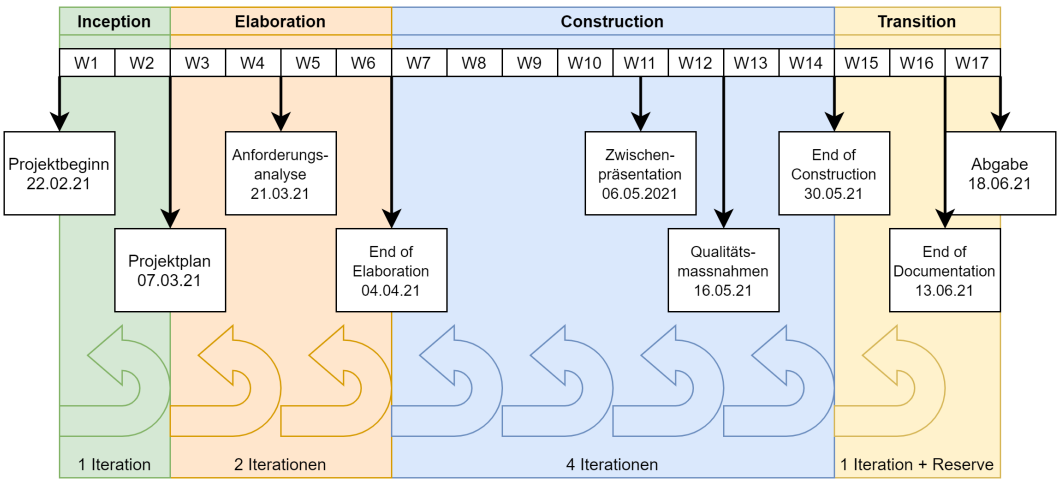

Abbildung 3.1: Übersicht Zeitplan

3.2 Phasen

Das Projekt wird nachfolgend, gemäss Projektmethodik aus Kapitel 4.3, in Phasen unterteilt. Das bedeutet konkret:

- Die Arbeit wird in 4 Phasen geplant.
- In jeder Phase gibt es Iterationen.
- Meilensteine werden auf das Ende von Iterationen gelegt.
- Es wird bei jeder Iteration (in der „Construction“-Phase) ein potentially shippable product angestrebt.
- Nach jeder Iteration findet eine Sprint-Review-Sitzung statt, um den Backlog mit Arbeitspaketen zu pflegen und das weitere Vorgehen zu besprechen.

- Nach jeder Iteration findet ein Austausch über das agile Vorgehen statt (Sprint Retrospective), um Probleme im Projektmanagement zu vermeiden.
- Es wird Wert auf ein sauberes End of Elaboration gelegt. Dazu wird die End of Elaboration Checklist verwendet.
- Am Ende der „Construction"-Phase gibt es einen Feature-Freeze.

3.2.1 Inception

- **Dauer:** 2 Wochen (+ Kickoff in Woche 0)
- **Zeitraum:** 22.02.21 - 07.03.21

Obwohl ein agiles Vorgehen für das Projekt angestrebt wird, wird das Projekt in dieser ersten Phase grob durchgeplant. Dies dient in erster Linie zur Aufteilung der begrenzten Projektzeit, um spätere Zeitengpässe zu vermeiden. Ausserdem ermöglicht der Projektplan, für alle Beteiligten einen gemeinsamen Ausgangspunkt zu schaffen und die Rahmenbedingungen zu definieren.

Des Weiteren werden die Anforderungen des Kunden aufgenommen und erstmalig spezifiziert. Die Spezifikation wird in einem weiteren Schritt mit dem Kunden besprochen.

3.2.2 Elaboration

- **Dauer:** 4 Wochen
- **Zeitraum:** 08.03.21 - 04.04.21

In einem ersten Schritt werden in dieser zweiten Projektphase, zusammen mit dem Kunden, die Anforderungen weiter ausgearbeitet. Durch die agile Natur des Projekts können diese sich im Laufe noch ändern, jedoch ist es sinnvoll die Bedürfnisse des Kunden möglichst genau zu erfassen, damit diese bei der anschliessenden Planung der Architektur auch berücksichtigt werden können.

Weiterhin wird die technische Machbarkeit des Projekts überprüft. Dies geschieht mittels eines minimalen Prototyps, welcher die wichtigsten Aspekte der geplanten Architektur implementiert. So sind per Ende der „Elaboration-Phase" alle bereits bekannten Unklarheiten beseitigt, was einen gut vorbereiteten Start der Implementation ermöglicht.

3.2.3 Construction

- **Dauer:** 8 Wochen
- **Zeitraum:** 05.04.21 - 30.05.21

Während dieser dritten und längsten Phase wird hauptsächlich das geplante Produkt umgesetzt. Parallel dazu, werden zur Gewährleistung der Code-Qualität Tests geschrieben und die Dokumentation nachgeführt.

In dieser Phase ist das Feedback des Kunden besonders wichtig, um das Endprodukt möglichst nach dessen Vorstellungen zu gestalten. Um dies zusätzlich überprüfen zu können werden Usability-Tests durchgeführt.

Am Ende der „Construction“-Phase findet ein Feature-Freeze statt. Dies bedeutet, dass keine neue Funktionalität mehr zum Produkt hinzugefügt wird.

3.2.4 Transition

- **Dauer:** 2 Wochen
- **Zeitraum:** 31.05.21 - 18.06.21

Diese letzte Phase dient zur Behebung verbleibender Fehler, zur Qualitätsprüfung und zur Übergabe des Produkts an dessen weiteren Betreiber. Des Weiteren wird das Projekt ausgewertet und letzte Verbesserungen an der Dokumentation vorgenommen, damit diese zur Abgabe bereit ist.

3.3 Iterationen / Sprints

Ein Sprint dauert stets zwei Wochen. Die Sprints beginnen jeweils am Montag und werden am Sonntag abgeschlossen. Ausgewertet werden die Sprints jeweils am zweiten Freitag mit anschliessender Planung des darauffolgenden Sprints. Die Auswertung am Freitag erlaubt es, allfällige noch nicht abgeschlossene Tätigkeiten bis Sonntag nachzuliefern.

Insgesamt werden in diesem Projekt 8 Sprints durchgeführt.

3.4 Meilensteine

Nachstehend werden die einzelnen Meilensteine spezifiert. Jeder dieser Meilensteine enthält Arbeitsprodukte, welche in die folgenden Kategorien eingeteilt werden:

- **Priorität 1:** Arbeitsprodukt, welches zwingend vor dem entsprechenden Meilenstein fertiggestellt werden muss.
- **Priorität 2:** Arbeitsprodukt, welches auch in der darauffolgenden Iteration beendet werden kann, sollte dies notwendig sein.

3.4.1 Projektplan

Ziel:	Projektplanung abgeschlossen
Zeitpunkt:	Ende Sprint 1 am 07.03.21
Arbeitsprodukte:	• Priorität 1 – Der Projektplan ist inklusive Projektorganisation, Zeitplan, Qualitätsmanagement und Risikoanalyse fertiggestellt. • Priorität 2 – keine

3.4.2 Anforderungsanalyse

Ziel:	Anforderungsanalyse abgeschlossen
Zeitpunkt:	Ende Sprint 2 am 21.03.21
Arbeitsprodukte:	• Priorität 1 – Use Cases sind im „Brief“-Stil verfasst. – Nicht-funktionale Anforderungen sind definiert. – Domainanalyse wurde durchgeführt. • Priorität 2 – Verwendete Lizenz ist definiert. – Schnittstellenbeschreibung ist erstellt. – UI-Skizzen sind erstellt.

3.4.3 End of Elaboration

Ziel:	Alle bekannten Unsicherheiten beseitigt
Zeitpunkt:	Ende Sprint 3 am 04.04.21
Arbeitsprodukte:	• Priorität 1 – End of Elaboration Checklist ist überprüft. – Architektur ist weitgehend spezifiziert. – Prototyp ist erstellt (technischer Durchstich). • Priorität 2 – Dokumentation ist auf dem neusten Stand.

3.4.4 Qualitätsmassnahmen

Ziel:	Produktqualität überprüft
Zeitpunkt:	Ende Sprint 6 am 16.05.21
Arbeitsprodukte:	• Priorität 1 – Wichtigste Funktionalitäten sind implementiert und getestet. • Priorität 2 – keine

3.4.5 End Of Construction

Ziel:	Geplante Funktionalität implementiert
Zeitpunkt:	Ende Sprint 7 am 30.05.21
Arbeitsprodukte:	• Priorität 1 – Alle geplanten Funktionalitäten sind implementiert. • Priorität 2 – Dokumentation ist auf dem neusten Stand.

3.4.6 End of Documentation

Ziel:	Dokumentation fertiggestellt
Zeitpunkt:	Ende Sprint 8 am 13.06.21
Arbeitsprodukte:	• Priorität 1 – Dokumentation ist komplett nachgeführt. – Das Abstract ist dem Betreuer abgegeben worden. • Priorität 2 – Layout- und Grammatikfehler sind nochmals überprüft worden.

3.4.7 Abgabe

Ziel:	Projekt abgeschlossen
Zeitpunkt:	Abgabetermin der Bachelorarbeit am 18.06.21
Arbeitsprodukte:	• Priorität 1 – Die Dokumentation wurde fertiggestellt. – Alle Dokumente wurden auf https://archiv.i.ost.ch/ hochgeladen. • Priorität 2 – Das Projekt wurde dem Kunden zum Betrieb übergeben.

4 Organisation

In diesem Kapitel wird die Organisation innerhalb der Projektes und die Verteilung der Aufgaben beleuchtet. Auch werden Vorgehensweisen wie die Projektmethodik und Projektabläufe definiert. Auch die Werkzeuge, welche für das Projektmanagement eingesetzt wurden, werden erwähnt.

4.1 Rollenverteilung

Das Projektteam besteht aus zwei Personen – Carlo Kirchmeier und Yannick Vogt. Mirko Stocker übernimmt die Betreuung und Bewertung, wobei er von Leo Büttiker als Experte und Laurent Metzger als Gegenleser unterstützt wird. Der Kunde, die Pronto AG, wird durch Ramon Herzog, Leiter Innovation und Entwicklung, sowie Heidi Herzog, Leiterin Administration vertreten. Heidi Herzog steht ausserdem bei Fragen zur IT Infrastruktur zur Verfügung.

Während der Arbeit wird es keine fixe Aufteilung der Aufgaben geben. Es wird allerdings aufgrund der jeweiligen Vorkenntnisse und Interessen der Teammitglieder so sein, dass Carlo Kirchmeier sich mehr auf das Backend und Yannick Vogt mehr auf das Frontend konzentriert. Der Austausch untereinander bleibt trotzdem im Fokus, was einen gleichwertigen Wissenstand über alle Aspekte des Projekts ermöglicht.

- **OST – Ostschweizer Fachhochschule**
  - Mirko Stocker – Betreuung / Bewertung
  - Leo Büttiker – Experte
  - Laurent Metzger – Gegenleser
  - Carlo Kirchmeier – Studierender
  - Yannick Vogt – Studierender
- **Pronto AG**
  - Ramon Herzog – Leiter Innovation und Entwicklung
  - Heidi Herzog – Leiterin Administration

4.2 Besprechungen

Besprechungen mit dem Betreuer werden vorraussichtlich wöchentlich abgehalten, jedoch wird dies nach Bedarf geregelt. Mit dem Kunden wird versucht, sich so oft wie möglich

und nötig auszutauschen, um ein zufriedenstellendes Endresultat zu erzielen. In beiden Fällen wird jedoch auf fixe Termine verzichtet.

Innerhalb des Projektteams gibt es am Ende jedes Sprints ein Review und Plannungsmeeting. Weitere Besprechungen werden, dank des kleinen Teams, spontan und der Notwendigkeit angepasst einberufen.

4.3 Projektmethodik

Es wurden diverse Projektmethodiken evaluiert. Genauer gesagt RUP, Wasserfall, Scrum und Scrum +. Anschliessend wurden Faktoren eruiert, welche die Wahl der Methodik beeinflussen:

1. Das Projektteam hat noch keine Erfahrungen in der Entwicklung von cross-platform Mobile Apps und kann sich dementsprechend nicht auf historische Daten beziehen. Solche Arbeiten auf der „grünen Wiese“ sind prädestiniert für einen agilen Ansatz.
2. Die Arbeit erfordert einiges an Exploration und ausprobieren, was ebenfalls für einen agilen Ansatz spricht.
3. Das Projektteam konnte bereits im Engineeringprojekt und der Studienarbeit Erfahrungen mit der Methodik Scrum + sammeln und fühlt sich daher mit dieser am sichersten.

Da die oben genanten Punkte einen agilen Ansatz nahelegen, sicherheitshalber ein End of Elaboration aber trotzdem als notwendig empfunden wird, fällt die Wahl auf Scrum +.

4.4 Projektabläufe

Während des ganzen Projekts finden zwei zentrale Abläufe statt. Zum einen ist dies das Sprint-Review und -Planning nach jeder Iteration und zum Anderen das Abarbeiten der Arbeitspakete.

4.4.1 Sprint-Review und Planning

Am Ende jeder Iteration erfolgt ein Sprint Review Meeting, bei welchem die Arbeitspakete des letzen Sprints überprüft werden. Konnten gewisse Arbeitspakete nicht abgeschlossen werden, oder wurden diese nach einem erneuten Review als nicht abgeschlossen angesehen, werden sie in den nächsten Sprint übernommen.

Sind alle Arbeitspakete besprochen, werden neue Arbeitspakete evaluiert, aus dem Backlog entnommen und für den nächsten Sprint eingeplant.

4.4.2 Arbeitspakete

Der Entwicklungsprozess sowie der Dokumentationsprozess basieren auf dem Feature Branch Workflow. Dabei wird sich an der Atlassian Feature Branch Workflow Dokumentation orientiert [2]. Es wird allerdings auf den Branch *develop* verzichtet. Der letzte stabile Release wird stattdessen durch einen Tag markiert.

Für den Workflow gelten folgende Regeln:

- Ex existiert ein Branch *master*, welcher als Integrationsbranch aller Feature-Branches fungiert.
  - Pro Version wird auf dem Branch *master* en Tag gesetzt und es findet ein neuer Release statt.
- Für jedes Feature wird ein eigener Feature-Branch erstellt.
  - Ein Feature-Branch wird ausgehend von einem Issue erstellt.
  - Ein Feature-Branch wird per Pull-Request in den Branch *master* integriert.

Ausserdem gelten spezifisch für Arbeitspakete folgende Regeln:

- Arbeitspakete werden als Issues erstellt.
- Die Zeitschätzung und Rapportierung erfolgt über Clockify.
- Nach Abschluss der zu erwartenden Arbeit im Issue wird ein Review gemacht.
- Es wird erst ein Tag gesetzt, wenn die Definition of Done für alle Arbeitspakete im Release, gemäss Qualitätsmanagement erreicht wurde.

Der folgende Ablauf gilt für Issues jeglicher Art, unabhängig davon, ob es sich dabei um Dokumentation oder Implementation handelt. Der Arbeitsschritt „Tests schreiben“ fällt für die Dokumentation weg:

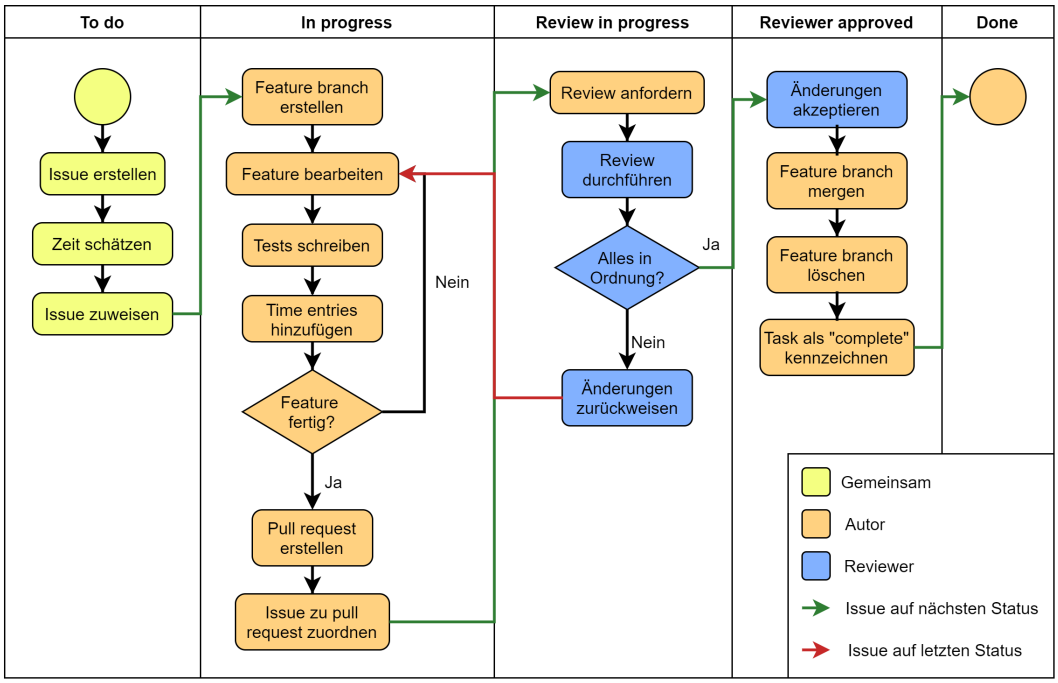

Abbildung 4.1: Lebenszyklus eines Arbeitspakets

4.5 Tools

Um das Projektmanagement zu erleichtern werden folgende Tools eingesetzt:

- GitHub:
  - Zur sauberen Aufteilung wird mit verschiedenen Repositories für Code und Dokumentation gearbeitet.
  - GitHub-Projects ermöglicht eine Übersicht über alle Issues in Form eines Kanban-Boards, aufgeteilt nach Status.
- LaTeX:
  - LaTeX wird verwendet, um die Dokumentation zu verfassen.
  - Der textbasierte LaTeX-Quellcode ermöglicht es, auch die Dokumentation sauber über Git zu versionieren.
- Clockify:
  - Clockify ermöglicht die Schätzung von Zeitaufwänden, deren Rapportierung und Auswertung.
  - Mithilfe einer Browsererweiterung lassen sich GitHub Issues direkt als Tasks in Clockify übertragen.
- Microsoft Teams:
  - Wird als primärer Kommunikationskanal innerhalb, aber auch ausserhalb des Projektteams genutzt.
  - Dient als Ersatz für persönliche Meetings während der Covid-19 Pandemie.

5 Qualitätsmanagement

In diesem Kapitel wird beschrieben welche Massnahmen ergriffen werden um die Qualität sowohl der Dokumentation als auch des Endproduktes zu garantieren.

5.1 Qualitätsmassnahmen

Zeitraum	Massnahme	Ziel
Freitags	Sprint Review	Austausch und Review über den Stand der Arbeitspakete.
Freitags	Sprint Planning	Neuen Sprint mit Arbeitspaketen aus Backlog planen. Falls nötig Priorisierungen anpassen.
Freitags	Meeting mit Projektbetreuung	Allfällige Unklarheiten klären, Projektstand besprechen.
Kontinuierlich	Continuous Integration	Durchführen von automatisierten Tests und statischer Code-Analyse.
Kontinuierlich	Codereview	Gegenseitiges Codereview bevor Features in den Master/Development-Branch integriert werden.
Kontinuierlich	Dokumentationsreview	Gegenseitiges Dokumentationsreview bevor Änderungen in den Master-Branch integriert werden.

Tabelle 5.1: Eingeplante Qualitätsmassnahmen

5.1.1 Projektmanagement

Die Basis für das Projektmanagement bildet ein Projektboard, welches über GitHub realisiert wird. Arbeitspakete werden als Issues angelegt und dem entsprechenden Meilenstein im Repository zugeordnet. Dies ermöglicht uns eine Übersicht über sämtliche Arbeitspakete und deren aktuellen Status. Die Zeiterfassung über das gesamte Projekt erfolgt über Clockify. Weitere Tools werden für das Projektmanagement vorerst nicht eingesetzt. Die Projektbetreuung hat Zugriff auf die GitHub Organisation und somit Zugriff auf sämtliche Repositories und den Stand der Arbeiten.

5.1.2 Dokumentation

Die Dokumentation befindet sich in einem eigenen Repository auf GitHub. Die Qualität bezüglich Inhalt, Formatierung und Grammatik wird mittels Peer-Reviews vor dem integrieren der Pull Requests gewährleistet.

5.1.3 Entwicklung

Der Source Code wird mit Git versioniert und befindet sich getrennt von der Dokumentation auf einem separaten Git-Repository auf GitHub.

Die Qualität des Source Codes wird, in einem ersten Schritt, automatisiert geprüft. Dies erfolgt im Rahmen einer Continuous Integration, durch einen Linter, gegen ein vordefiniertes Stylesheet. Des Weiteren werden in dieser Continuous Integration auch Unit und Integration Tests durchgeführt. In einem letzten Schritt wird die Codequalität zusätzlich durch ein Peer-Review der Pull Requests auf GitHub geprüft.

5.2 Definition of Done

Um Unklarheiten bezüglich der Vollständigkeit von Arbeitspaketen, Sprints und Meilensteine zu beseitigen, wird für diese in den folgenden Abschnitten eine Definition of Done festgelegt.

5.2.1 Arbeitspaket

- Automatisierte Tests in der Continuous Integration zeigen keine Fehler an, allfällige Warnungen wurden überprüft und abgearbeitet.
- Die GitHub Action-Pipeline ist erfolgreich durchlaufen.
- Sämtliche Unit Tests wurden fehlerfrei ausgeführt.
- Die nicht-funktionalen Anforderungen sind/bleiben erfüllt.
- Es wurde ein Review des Codes beziehungsweise der Dokumentationsänderungen durchgeführt.

5.2.2 Sprint

- Nicht abgeschlossene Arbeitspakete in den nächsten Sprint verschoben
- Definition of Done für alle verbleibenden Arbeitspakete im Sprint erreicht
- Sprint Review durchgeführt

5.2.3 Meilenstein

- Definition of Done für alle Sprints im Meilenstein erreicht
- Die geplanten Resultate des Meilensteins fertiggestellt

5.3 Testing

In diesem Kapitel wird beschrieben, welche Arten von Tests durchgeführt werden. Auch wird definiert wann diese Tests gemacht und wie diese Dokumentiert werden.

5.3.1 Unit und Integration Tests

Um eine gute Testabdeckung zu garantieren, wird explizit Zeit eingeplant, um Tests zu schreiben. Diese werden im Review auch angeschaut und auf ihre Tauglichkeit geprüft. Priorisiert wird der anwendungskritische Code, konkret die Businesslogik.

Trivialer Code, wie beispielsweise Getter und Setter, wird nicht getestet. Es kann davon ausgegangen werden, dass bei Fehlern in diesem Code Folgefehler auftreten. Diese werden in weiteren Tests abgefangen. Ausserdem wird generell auf Tests für Code verzichtet, der lediglich als Bindeglied zwischen zwei Bibliotheken fungiert.

5.3.2 Systemtests

Anhand der definierten Use-Cases werden die Systemtests durchgeführt und ausgewertet. Dies wird regelmässig manuell von den Teammitgliedern gemacht.

5.3.3 Nichtfunktionale Tests

Basierend auf den nicht-funktionalen Anforderungen werden geeignete nicht-funktionale Tests durchgeführt. Bereits geplant ist ein Usability-Test am Ende des Qualitätsmassnahmen-Meilensteins.

5.3.4 Abnahmetests

Es ist ein Abnahmetest mit dem Endkunden vorgesehen. Dieser wird voraussichtlich an einer der letzten Sitzungen mit dem Kunden durchgeführt.

6 Risikomanagement

Arbeitspakete werden gemäss ihrem Risiko priorisiert. Je grösser das Risiko, dass ein Arbeitspaket scheitert, desto früher wird es erledigt. So soll möglichst früh ein „Durchstich“ durch alle technischen Schichten erfolgen, damit technische Probleme frühzeitig erkannt werden. Am Schluss eingeplant sind „Nice to have“- Funktionalitäten, wie das Aufteilen der Einsatzpläne nach Abteilung oder der Bereich für Schulungsvideos, da diese bei Zeitknappheit weggelassen werden können.

Es werden nur wenige zeitliche Reserven gebildet – stattdessen ist das Projekt so geplant, dass möglichst früh ein funktionierender Prototyp vorhanden ist, der im weiteren Verlauf optimiert und ausgebaut wird. So kann auch dann ein lauffähiges Produkt ausgeliefert werden, wenn zeitlich bedingt nicht alle Arbeitspakete abgeschlossen werden konnten.

Als minimaler Buffer ist die Zeit zwischen dem Meilenstein „Qualitätsmassnahmen“ und dem Meilenstein „End of Construction“ vorgesehen. Ausserdem ist die letzte Woche der Bachelorarbeit für abschliessende Arbeiten vorgesehen und kann somit als zusätzlicher Buffer dienen.

6.1 Risiken

In diesem Kapitel geht es um die evaluierten Risiken und die Abschätzung, welchen Einfluss diese auf den Verlauf der Arbeit haben könnten.

6.1.1 Risiko: Ausfall eines Teammitgliedes

Da das Team nur aus zwei Personen besteht, hat der Ausfall eines Mitglieds einen starken Einfluss auf den Verlauf des Projektes. Gerade zu Zeiten von Covid ist es möglich, dass ein Mitglied krankheitshalber ausfällt.

- **Auswirkungen:** Der Zeitplan kann je nach Schweregrad der Erkrankung nicht mehr eingehalten werden. Im Falle einer Quarantäne kann zwar weitergearbeitet werden, die Kommunikation wäre allerdings erschwert.
- **Maximaler Schaden:** 36h
- **Eintrittswahrscheinlichkeit:** 30%
- **Gewichteter Schaden:** 10.8h
- **Vorbeugung:** Durch regelmässige Absprachen im Team wird sichergestellt, dass alle Teammitglieder auf demselben Informationsstand sind. Dadurch sind bei einem Ausfall alle nötigen Informationen zur Forsetzung der Arbeit vorhanden.

- **Verhalten beim Eintreten:** Die erkrankte Person arbeitet in dem Mass weiter, wie es ihr möglich ist. Die andere Person arbeitet normal weiter. Gegebenenfalls muss mit dem Kunden eine Reduktion des Umfangs angeschaut werden, um den Ausfall zu kompensieren.

6.1.2 Risiko: Missverständnisse in der Kommunikation

In diesem Projekt arbeiten einige Parteien zusammen. Aufgrund dessen, kann es zu Missverständnissen und Fehlinterpretationen kommen, welche Zeit kosten.

- **Auswirkungen:** Je nach Missverständnis muss im schlimmsten Fall ein ganzer Anwendungsfall neu implementiert werden.
- **Maximaler Schaden:** 48h
- **Eintrittswahrscheinlichkeit:** 20%
- **Gewichteter Schaden:** 9.6h
- **Vorbeugung:** Die Anwendungsfälle werden früh spezifiziert und mit dem Kunden abgesprochen. Während der Umsetzung wird regelmässig Feedback vom Kunden eingeholt.
- **Verhalten beim Eintreten:** Die Betreuungsperson sowie der Kunde werden über das Missverständnis informiert. Anschliessend wird besprochen wie weiter vorgegangen werden soll.

6.1.3 Risiko: Implementation Push-Benachrichtigungen problematisch

Um Mitarbeitern zeitnah neue Einsatzpläne und Nachrichten zukommen zu lassen, werden Push-Benachrichtigungen benötigt. Um diese implementieren zu können, muss es möglich sein, diese in das gewählte App-Framework zu integrieren. Es wäre denkbar, dass dies beim gewählten Framework schwierig oder gar unmöglich ist.

- **Auswirkungen:** Falls die Integration unmöglich sein sollte, müsste ein neues Framework evaluiert und anschliessend verwendet werden. Falls die Integration nur schwierig möglich ist, muss entschieden werden, ob diese sich lohnt oder ein Wechsel des Frameworks nachhaltiger wäre.
- **Maximaler Schaden:** 48h
- **Eintrittswahrscheinlichkeit:** 10%
- **Gewichteter Schaden:** 4.8h
- **Vorbeugung:** Bei der Evaluation des Frameworks, zu Beginn des Projektes, wird bewusst darauf geachtet, dass Push-Benachrichtigungen unterstützt werden und sich der Integrationsaufwand in einem machbaren Rahmen hält.
- **Verhalten beim Eintreten:** Die Betreuungsperson wird informiert und anschliessend der entstehende Aufwand abgeschätzt. Je nach entstehendem Aufwand muss mit dem Kunden eine Reduktion des Umfangs angeschaut werden.

6.1.4 Risiko: Serverressourcen eingeschränkt

Für die Umsetzung der Arbeit und die Datenhaltung wird ein Server benötigt, auf welchem ein Teil der Applikation läuft. Die Art des Servers hängt vom Angebot des Kunden respektive dessen Anbieter ab.

- **Auswirkungen:** Der Serverteil der Applikation muss angepasst werden, da dieser nicht mit dem vom Kunden zur Verfügung gestellten Server kompatibel ist.
- **Maximaler Schaden:** 36h
- **Eintrittswahrscheinlichkeit:** 5%
- **Gewichteter Schaden:** 1.8h
- **Vorbeugung:** Es wird möglichst früh zusammen mit dem Kunden abgeklärt, welches Budget und welche Möglichkeiten im Bezug auf den Server vorhanden sind.
- **Verhalten beim Eintreten:** Der Betreuer sowie der Kunde wird informiert. Es wird abgeklärt ob der Server entsprechend der Applikation angepasst werden kann. Sollte dies nicht möglich sein, so wird die Applikation umgeschrieben.

6.1.5 Risiko: iOS Build problematisch

Das Projektteam benutzt privat keine Geräte der Marke Apple und hat somit wenig Erfahrung mit sowohl iOS als auch macOS. Erschwerend kommt hinzu, dass für die Entwicklung von iOS-Apps zwingen ein Apple-Gerät verwendet werden muss.

- **Auswirkungen:** Falls kein Apple-Gerät organisiert werden kann, so muss entweder ein Build-Service eingekauft oder auf die Integration in iOS verzichtet werden. Alternativ ist es möglich, dass durch die fehlende Erfahrung des Projektteams Mehraufwand bei der Integration in iOS entsteht.
- **Maximaler Schaden:** 16h
- **Eintrittswahrscheinlichkeit:** 10%
- **Gewichteter Schaden:** 1.6h
- **Vorbeugung:** Der Betreuer ist bereits daran ein Apple-Gerät für die Entwicklung zu organisieren und eine Developer-Lizenz konnte bereits organisiert werden. Bei der Evaluation des Frameworks wird ausserdem darauf geachtet, dass die Integration in iOS möglichst ohne MacOS-Spezifische Sonderschritte durchführbar ist.
- **Verhalten beim Eintreten:** Falls es sich lediglich um Mehraufwand durch fehlende Erfahrung handelt, wird dieser einfach verbucht. Sollte es nicht möglich sein ein Gerät aufzutreiben, muss mit dem Kunden angeschaut werden, ob dieser auf iOS verzichten kann oder ein Service/Gerät eingekauft werden soll.

6.2 Erkenntnisse

Der maximale gewichtete Schaden beträgt 28.6h. Aufgrund der frühen Erkennung von Problemen, wie in Kapitel 6 beschrieben, sollte es allerdings kein Problem sein, diesen Schaden mittels einer Anpassung des Projektplans und der allfälligen Streichung von optionalen Features abzufedern. Falls im schlimmsten Fall mehrere/alle Risiken eintreffen, muss ein Treffen mit der Betreuungsperson und dem Kunden gehalten werden um das Projekt neu zu evaluieren.

Teil II

Vorstudie

7 | Übersicht

Im Teil „Vorstudie“ werden die für die Umsetzung der Arbeit notwendigen Informationen zusammengetragen. Ausserdem werden Anforderungen, welche an das Endergebnis gestellt werden, definiert. Zu guter Letzt werden die Herausforderungen, welche erkannt wurden, spezifiziert, analysiert und die daraus entstandenen Entscheidungen dokumentiert.

8 Anforderungsspezifikationen

In diesem Kapitel geht es darum, die Anforderungen an die Applikation aufzunehmen und diese anschliessend zu spezifizieren, sowie zu dokumentieren. Hierbei ist eine gute Zusammenarbeit mit dem Kunden sehr wichtig, denn nur so können Missverständnisse und daraus resultierende Anpassungen vermieden werden.

8.1 Lizenzierung

Aufgrund der Tatsache, dass der Kunde keine Anforderungen an die Lizenzierung stellt, ist das Entwicklerteam in deren Wahl frei. Die Entscheidung wurde dementsprechend innerhalb des Entwicklerteams besprochen und es wurde die MIT-Lizenz gewählt. Diese Lizenz ist sehr offen und entwicklerfreundlich. Dadurch wird verhindert, dass Lizenzbedingungen allfällig verwendeter Software-Bibliotheken verletzt werden.

Es wurde ebenfalls entschieden, den Quellcode öffentlich einsehbar zu machen, damit andere Firmen oder Privatpersonen auch von diesem profitieren können. Auch können dadurch etwaige Fehler mithilfe der Community schneller erkannt werden und es stehen den Entwicklern mehr Tools ohne Zusatzkosten zur Verfügung.

8.2 Funktionale Anforderungen (Use Cases)

In diesem Kapitel werden die funktionalen Anforderungen an die Applikation definiert. Zur Veranschaulichung der, mit dem Kunden erarbeiteten, funktionalen Anforderungen, wurden diese in Abbildung 8.1, zusammen mit den Betroffenen Aktoren, grafisch dargestellt. Die in der Grafik dargestellten Aktoren, werden in Kapitel 8.2.1 noch genauer spezifiziert.

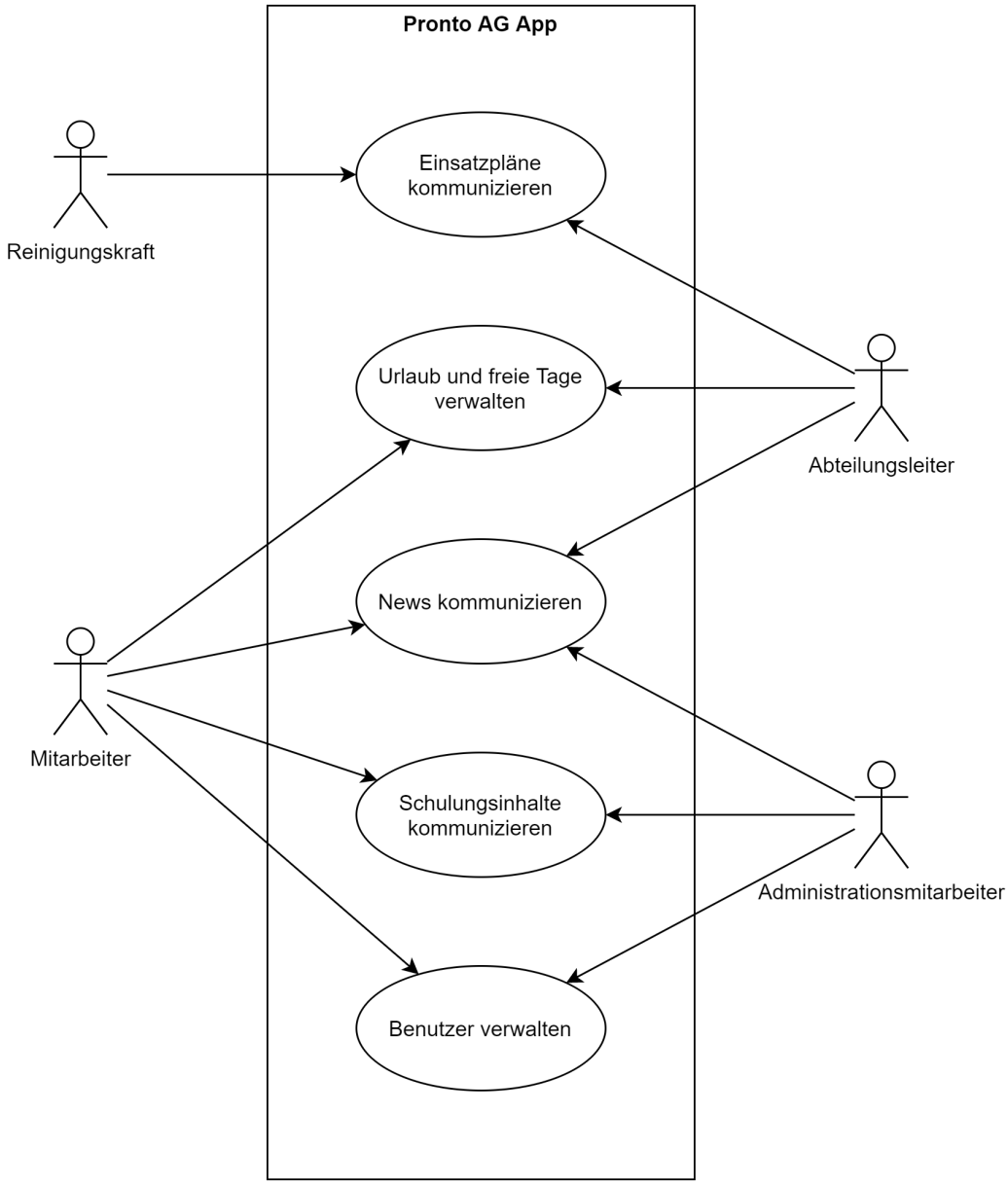

Abbildung 8.1: Übersicht Use Cases

8.2.1 Aktoren

Mitarbeiter	Der Mitarbeiter fungiert als Überbegriff für alle anderen genannten Aktoren. Zu seinen applikationsrelevanten Tätigkeiten gehört die Beantragung von Urlaub und freien Tagen, sowie der Konsum von Schulungsinhalten und News.
Reinigungskraft	Die Reinigungskraft ist zuständig für die Abarbeitung der Termine gemäss Einsatzplan. Zu den applikationsrelevanten Tätigkeiten gehört der Abruf von Einsatzplänen.
Abteilungsleiter	Der Abteilungsleiter leitet ein Team an Reinigungskräften. Zu den applikationsrelevanten Tätigkeiten des Abteilungsleiters gehört die Verwaltung von Einsatzplänen, das Genehmigen von Urlaub und freien Tagen, sowie das Publizieren von abteilungsspezifischen News.
Administration	Der Administrationsmitarbeiter ist für die Verwaltung von News, Schulungsinhalten und Mitarbeitern zuständig. Zu den applikationsrelevanten Tätigkeiten des Administrationsmitarbeiters gehört die Verwaltung eben dieser Daten.

8.2.2 UC01 – Einsatzpläne kommunizieren

Die Applikation soll einen Bereich bieten, um Einsatzpläne kommunizieren zu können. Dies beinhaltet die Erstellung dieser Einsatzpläne durch die Administration und deren Einsicht durch die Reinigungskräfte.

8.2.2.1 US01.01 – Einsatzpläne verwalten

Als Abteilungsleiter möchte ich Einsatzpläne erstellen, bearbeiten und entfernen können, um den Reinigungskräften bevorstehende Aufgaben mitzuteilen.

Ausbaustufe – Dateien
Die Einsatzpläne können als PDF-Dateien hochgeladen werden.

Ausbaustufe – Integriert
Einsatzpläne können direkt innerhalb der Applikation mittels editierbarer Tabellen verwaltet werden.

8.2.2.2 US01.02 – Einsatzpläne publizieren

Als Administrationsarbeiter möchte ich zuvor erstellte Einsatzpläne publizieren können, um diese den Mitarbeitern zugänglich zu machen.

Ausbaustufe – Firma
Einsatzpläne können auf Firmenebene publiziert werden.

Ausbaustufe – Abteilungsspezifisch
Einsatzpläne können auf Abteilungsebene publiziert werden.

Ausbaustufe – Benachrichtigung
Beim Publizieren der Einsatzpläne werden die entsprechenden Reinigungskräfte per Push-Benachrichtigung informiert.

8.2.2.3 US01.03 – Einsatzpläne abrufen

Als Reinigungskraft möchte ich meine Einsatzpläne abrufen können, um mich über bevorstehende Termine zu informieren.

Ausbaustufe – Dateien
Die Einsatzpläne stehen als PDF-Dateien zum Download über die Applikation bereit.

Ausbaustufe – Integriert
Die Einsatzpläne können direkt innerhalb der Applikation abgerufen werden.

Ausbaustufe – Kalender
Die Einsatzpläne können im Kalender des jeweiligen Geräts integriert werden.

8.2.3 UC02 – Urlaub und freie Tage verwalten

Mithilfe der Applikation soll es möglich sein, Urlaub sowie freie Tage zu verwalten und zu beantragen.

8.2.3.1 US02.01 – Jahresurlaub erfassen

Als Mitarbeiter möchte ich meinen Jahresurlaub erfassen und beantragen können, um anschliessend meine private Ferienplanung durchführen zu können.

Ausbaustufe – Einfach
Ein Mitarbeiter kann unlimitiert Urlaub erfassen.

Ausbaustufe – Begrenzt
Die Anzahl Urlaubstage, sowie die Länge zusammenhängender Urlaube kann definiert werden.

8.2.3.2 US02.02 – Jahresurlaub genehmigen

Als Abteilungsleiter möchte ich den Urlaub meiner Mitarbeiter genehmigen können, um den Urlaubsplan der Abteilung zu koordinieren.

Ausbaustufe – Einfach
Der Abteilungsleiter kann die Urlaube der Mitarbeiter genehmigen.

8.2.3.3 US02.03 – freie Tage erfassen

Als Mitarbeiter möchte ich freie Tage erfassen und beantragen können, um anschliessend diese Tage in Anspruch zu nehmen.

Ausbaustufe – Einfach
Ein Mitarbeiter kann unlimitiert freie Tage erfassen.

Ausbaustufe – Begrenzt
Die Beantragung ist nur bis zu einem definierten Abstand vor dem Datum des freien Tages möglich.

Ausbaustufe – Zeitbasiert
Überstunden sowie verbleibender Urlaub wird in die Beantragung integriert.

8.2.3.4 US02.04 – freie Tage genehmigen

Als Abteilungsleiter möchte ich freie Tage für meine Mitarbeiter genehmigen können, um die Einsatzplanung entsprechend anzupassen.

Ausbaustufe – Einfach
Der Abteilungsleiter kann die freien Tage der Mitarbeiter genehmigen.

8.2.3.5 US02.05 – Persönliche Übersicht freie Tage einsehen

Als Mitarbeiter möchte ich meine beantragten und die bewilligten Urlaube sowie freien Tage einsehen können, um über diese auf dem laufenden zu sein.

Ausbaustufe – Liste
Die Einträge werden in einer Liste angezeigt.

Ausbaustufe – Kalender
Die Einträge werden in Kalenderform dargestellt.

Ausbaustufe – Integriert
Die Einträge werden im Kalender des Geräts integriert.

8.2.3.6 US02.06 – Abteilungs-Übersicht freie Tage einsehen

Als Abteilungsleiter möchte ich die beantragten und die bewilligten Urlaube sowie freien Tage meiner Abteilung einsehen können, um über diese auf dem laufenden zu sein.

Ausbaustufe – Liste
Die Einträge werden in einer Liste angezeigt.

Ausbaustufe – Kalender
Die Einträge werden in Kalenderform dargestellt.

Ausbaustufe – Integriert
Die Einträge werden im Kalender des Geräts integriert.

8.2.4 UC03 – Schulungsinhalte kommunizieren

Innerhalb der Applikation soll ein Schulungsbereich für Mitarbeiter entstehen, welcher diesen die Möglichkeit gibt, sich eigenständig in gewissen Themenbereichen zu vertiefen.

8.2.4.1 US03.01 – Schulungsvideos Abrufen

Als Mitarbeiter möchte ich Schulungsvideos abrufen können, um mich über Neuerungen auf dem Laufenden zu halten.

Ausbaustufe – Dateien
Es stehen Videodateien zum Download über die Applikation bereit.

Ausbaustufe – Integriert
Videos können direkt innerhalb der Applikation abgerufen werden.

Ausbaustufe – Ausführlich
Zusätzlich zu den Videos werden noch Erklärungen und ergänzende Informationen angezeigt.

8.2.4.2 US03.02 – Schulungsvideos verwalten

Als Administrationsmitarbeiter möchte ich Schulungsvideos erstellen, bearbeiten und entfernen können, um den Mitarbeitern stets aktuelles Fortbildungsmaterial zu liefern.

Ausbaustufe – Dateien
Videos können als Datei hochgeladen werden.

Ausbaustufe – Ausführlich
Zusätzlich zu den Videos können noch Erklärungen und ergänzende Informationen definiert werden.

8.2.4.3 US03.03 – Anleitungen abrufen

Als Reinigungskraft möchte ich Anleitungen abrufen können, um mich über die auszuführenden Arbeitsschritte zu informieren.

Ausbaustufe – Dateien
Es Stehen PDF-Dateien zum Download über die Applikation bereit.

Ausbaustufe – Integriert
Anleitungen können direkt innerhalb der Applikation abgerufen werden.

8.2.4.4 US03.04 – Anleitungen Verwalten

Als Administrationsmitarbeiter möchte ich Anleitungen erstellen, bearbeiten und entfernen können, um den Mitarbeitern die notwendigen Arbeitsschritte verschiedener Arbeiten einfach zugänglich zu machen.

Ausbaustufe – Dateien
Anleitungen können als PDF-Dateien hochgeladen werden.

Ausbaustufe – Integriert
Anleitungen können direkt innerhalb der Applikation mittels WYSIWYG-Editor verwaltet werden.

8.2.5 UC04 – News kommunizieren

Im Rahmen der Applikation soll ein Newsportal entwickelt werden, über welches Neuigkeiten geteilt und abgerufen werden können.

8.2.5.1 US04.01 – News abrufen

Als Reinigungskraft möchte ich Neuigkeiten und Infos der Firma abrufen können, um stets auf dem neusten Stand zu sein und entsprechend reagieren zu können.

Ausbaustufe – Integriert
News können direkt innerhalb der Applikation angeschaut werden.

Ausbaustufe – Firma
News können auf Firmenebene abgerufen werden.

Ausbaustufe – Abteilungsspezifisch
News können auf Abteilungsstufe abgerufen werden.

Ausbaustufe – Hierarchie
News können von Mitarbeitern gemäss der definierten Hierarchie abgerufen und angeschaut werden.

8.2.5.2 US04.02 – News verwalten

Als Administrationsmitarbeiter möchte ich Neuigkeiten erstellen, bearbeiten und entfernen können, um diese anschliessend den Mitarbeitern zur Verfügung zu stellen.

Ausbaustufe – Dateien
News können als PDF-Dateien hochgeladen werden.

Ausbaustufe – Integriert
News können direkt innerhalb der Applikation mittels WYSIWYG-Editor verwaltet werden.

Ausbaustufe – Firma
News können auf Firmenebene erstellt werden.

Ausbaustufe – Abteilungsspezifisch
News können auf Abteilungsstufe erstellt werden.

Ausbaustufe – Abteilungsleiter
News auf Abteilungsstufe können vom Abteilungsleiter erstellt werden.

Ausbaustufe – Hierarchie
News können an die, in der Hierarchie definierten, Funktionen zugewiesen werden.

8.2.5.3 US04.03 – News publizieren

Als Administrationsmitarbeiter möchte ich Neuigkeiten publizieren können, um diese den Mitarbeitern zugänglich zu machen.

Ausbaustufe – Firma
News können auf Firmenebene publiziert werden.

Ausbaustufe – Abteilungsspezifisch
News können auf Abteilungsstufe publiziert werden.

Ausbaustufe – Abteilungsleiter
News auf Abteilungsstufe können vom Abteilungsleiter publiziert werden.

Ausbaustufe – Hierarchie
News können für die, in der Hierarchie definierten, Hierarchiestufen publiziert werden.

8.2.6 UC05 – Benutzer verwalten

Um die anderen Funktionalitäten der Applikation umsetzen zu können, ist eine Benutzerverwaltung notwendig, welche die von den Funktionen benötigte Firmenstruktur abbildet

und die Authentisierung ermöglicht.

8.2.6.1 US05.01 – Benutzer verwalten

Als Administrationsmitarbeiter möchte ich Mitarbeiter erstellen, bearbeiten und entfernen können um die Organisationsstruktur korrekt abzubilden.

Ausbaustufe – Abteilungsleiter
Für Mitarbeiter können deren Vorgesetzte definiert werden.

Ausbaustufe – Berechtigungen
Bei jedem Benutzer können die Berechtigungen angegeben werden, welche er innerhalb der Applikation hat.

Ausbaustufe – Hierarchie
Jeder Benutzer kann einer Hierarchiestufe zugewiesen werden.

8.2.6.2 US05.02 – Profil bearbeiten

Als Mitarbeiter möchte ich mein eigenes Profil bearbeiten können, um meine Informationen auf dem neusten Stand zu halten.

Ausbaustufe – Passwort
Das Passwort kann vom Mitarbeiter zurückgesetzt/angepasst werden.

8.3 Nicht-funktionale Anforderungen

Die Anforderungen heissen „nicht funktional“, da Sie nicht direkt mit der Erfüllung des Auftrages der Software zu tun haben, sondern mit deren Qualität. Die nicht funktionalen Anforderungen wurden mithilfe des FURPS-Model definiert. FURPS selbst ist ein Akronym aus dem Englischen und steht für die Softwarequalitäts-Merkmale Funktionalität, Benutzbarkeit, Zuverlässigkeit, Effizienz und Änderbarkeit. Es ist daher bei jedem NFR angegeben, in welche Kategorie des Akronyms es fällt. Die deutschen Begriffe der Kategorien wurden ISO/IEC 9126-1:2001 entnommen [3].

8.3.1 NFR01

- **Kategorie:** Funktionalität
- **Beschreibung:** Daten können erst nach erfolgreicher Anmeldung ausgelesen werden.

8.3.2 NFR02

- **Kategorie:** Benutzbarkeit
- **Beschreibung:** Ein Benutzer soll alle Aufgaben welche, gemäss den Use Cases aus Kapitel 8.2, dem Aktor „Mitarbeiter“ oder „Reinigungskraft“ zugeordnet sind, selbständig ausführen können. Hierbei soll keine zusätzliche Hilfestellung von aussen notwendig sein.

8.3.3 NFR03

- **Kategorie:** Benutzbarkeit
- **Beschreibung:** Informationen die farbcodiert sind, sollen durch eine zusätzliche Methode dargestellt werden, um farbenblinde Personen zu unterstützen.

8.3.4 NFR04

- **Kategorie:** Benutzbarkeit
- **Beschreibung:** Da nicht alle Benutzer gut Deutsch können, müssen Informationen auch visuell und nicht nur textuell dargestellt werden. Ausnahmefälle sind zu begründen.

8.3.5 NFR05

- **Kategorie:** Zuverlässigkeit
- **Beschreibung:** Tritt unerwartetes Verhalten auf, so wird dem Benutzer eine Fehlermeldung angezeigt. Dies wird mithilfe einer globalen Fehlerbehandlung bewerkstelligt.

8.3.6 NFR06

- **Kategorie:** Änderbarkeit (Wartbarkeit)
- **Beschreibung:** Klassen sind nicht länger als 300 Zeilen.

8.3.7 NFR07

- **Kategorie:** Änderbarkeit (Wartbarkeit)
- **Beschreibung:** Methoden sind nicht länger als 30 Zeilen.

8.3.8 NFR08

- **Kategorie:** Änderbarkeit (Wartbarkeit)
- **Beschreibung:** Zeilen sind nicht länger als 80 Zeichen.

8.3.9 NFR09

- **Kategorie:** Änderbarkeit (Wartbarkeit)
- **Beschreibung:** Der McCabe-Wert jeder Methode in der Codebase liegt unter 10.

8.3.10 NFR10

- **Kategorie:** Änderbarkeit (Wartbarkeit)
- **Beschreibung:** Keine Methode in der Codebase hat mehr als 5 Argumente.

8.3.11 NFR11

- **Kategorie:** Änderbarkeit (Wartbarkeit)
- **Beschreibung:** Die Testabdeckung des Codes beträgt mindestens 80%.

8.3.12 NFR12

- **Kategorie:** Änderbarkeit (Wartbarkeit)
- **Beschreibung:** Alle möglichen Aufrufszenarien der API werden dokumentiert.

8.3.13 NFR13

- **Kategorie:** Änderbarkeit (Wartbarkeit)
- **Beschreibung:** Es wird eine Installationsanleitung, sowie eine Wartungsanleitung erstellt.

8.4 Modellüberlegungen

In diesem Kapitel wird versucht ein erstes Modell zu erstellen, welches die benötigten Funktionalitäten, welche vom Kunden gewünscht wurden, abbilden kann. Hierzu wurde eine erste Skizze des Domain-Models erstellt, welche alle benötigten Business-Objekte abbildet. Auch werden in diesem Kapitel erste Überlegungen zur Benutzeroberfläche gemacht und mit Wireframes grafisch abgebildet.

8.4.1 Domain-Model

Im Domain-Model der Problemdomain wurden alle für die, vom Kunden gewünschte, Funktionalität benötigten Business-Objekte abgebildet. Es wurde ausserdem jedes Objekt noch textuell erläutert.

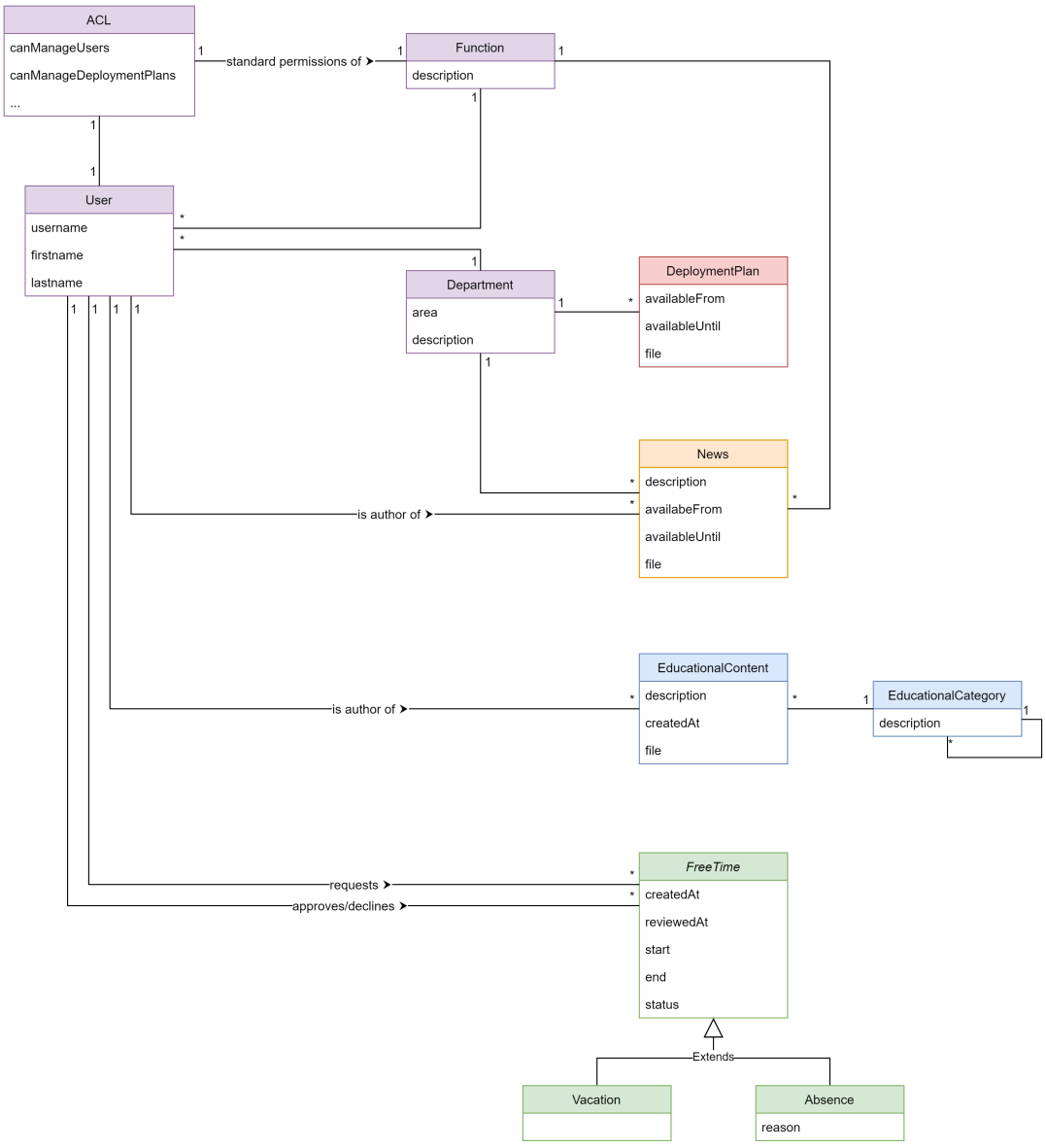

Abbildung 8.2: Übersicht über die Problemdomain

ACL	Beschreibt eine Liste von Berechtigungen, welche einem Benutzer zugewiesen sind.
Function	Beschreibt eine Funktion, welche der Benutzer innehat und seine Berechtigungen beeinflusst z.B. Reinigungskraft.
User	Beschreibt einen Benutzer, welcher die Applikation nutzt.
Department	Beschreibt eine Abteilung, welcher Benutzer, Einsatzpläne und News zugeordnet sind.
DeploymentPlan	Beschreibt einen Einsatzplan, welcher von Benutzern abgerufen werden kann.
News	Beschreibt einen News-Eintrag, welcher von Benutzern abgerufen werden kann.
EducationalContent	Beschreibt eine Schulungsunterlage, welche einer bestimmten Kategorie angehört.
EducationalCategory	Beschreibt eine Kategorie, welcher Schulungsunterlagen zugeordnet werden können.
FreeTime	Beschreibt freie Zeit, welche ein Benutzer beantragt hat.
Vacation	Beschreibt einen Urlaub, welcher ein Benutzer beantragt hat.
Absence	Beschreibt eine Absenz, welche ein Benutzer beantragt hat.

8.4.2 Benutzeroberfläche

Die Benutzeroberfläche ist, dass für den Kunden wahrscheinlich wichtigste Element der Applikation. Daher muss diese gut geplant und sorgfältig aufgebaut werden. Nachfolgend sind die Punkte dokumentiert, welche während der Planung der Benutzeroberfläche relevant waren.

8.4.2.1 Zusätzliche Webseite

Nach Absprache mit dem Kunden, wird neben dem geforderten cross-platform App auch eine Webseite, welche primär für die administrativen Funktionalitäten gedacht ist, entwickelt.

Aus der Framework Evaluation aus Kapitel 9 hat sich ergeben, dass die Wahl des cross-platform-Frameworks auf Flutter fällt. Da Flutter neben Android und iOS auch Support für Web bietet, wird angestrebt, sämtliche Funktionalitäten auch für das Web zur Verfügung zu stellen.

8.4.2.2 Design

Massgebend für das Design der Benutzeroberfläche sind die nicht-funktionalen Anforderungen in der Kategorie Usability, sowie das Corporate Design der Pronto AG.

Sollten zur Hervorhebung oder Kategorisierung (z.B. Info- / Warn- und Fehlermeldungen) weitere Farben benötigt werden, wird auf die Farbpalette des verwendeten UI-Frameworks zurückgegriffen.

8.4.2.3 Wireframes

Zur verbesserten Veranschaulichung der funktionalen Anforderungen für den Kunden, wurden Wireframes erstellt. Des Weiteren fungieren die Wireframes als Implementationsvorlage für die Benutzeroberfläche.

Die wichtigsten Funktionalitäten für Reinigungskräfte und allgemeine Mitarbeiter sind im Format eines Smartphone Apps abgebildet. Für die administrativen Funktionalitäten, welche primär von Abteilungsleitern und Administrationsmitarbeitern verwendet werden, wurden Wireframes im Desktop-Format erstellt.

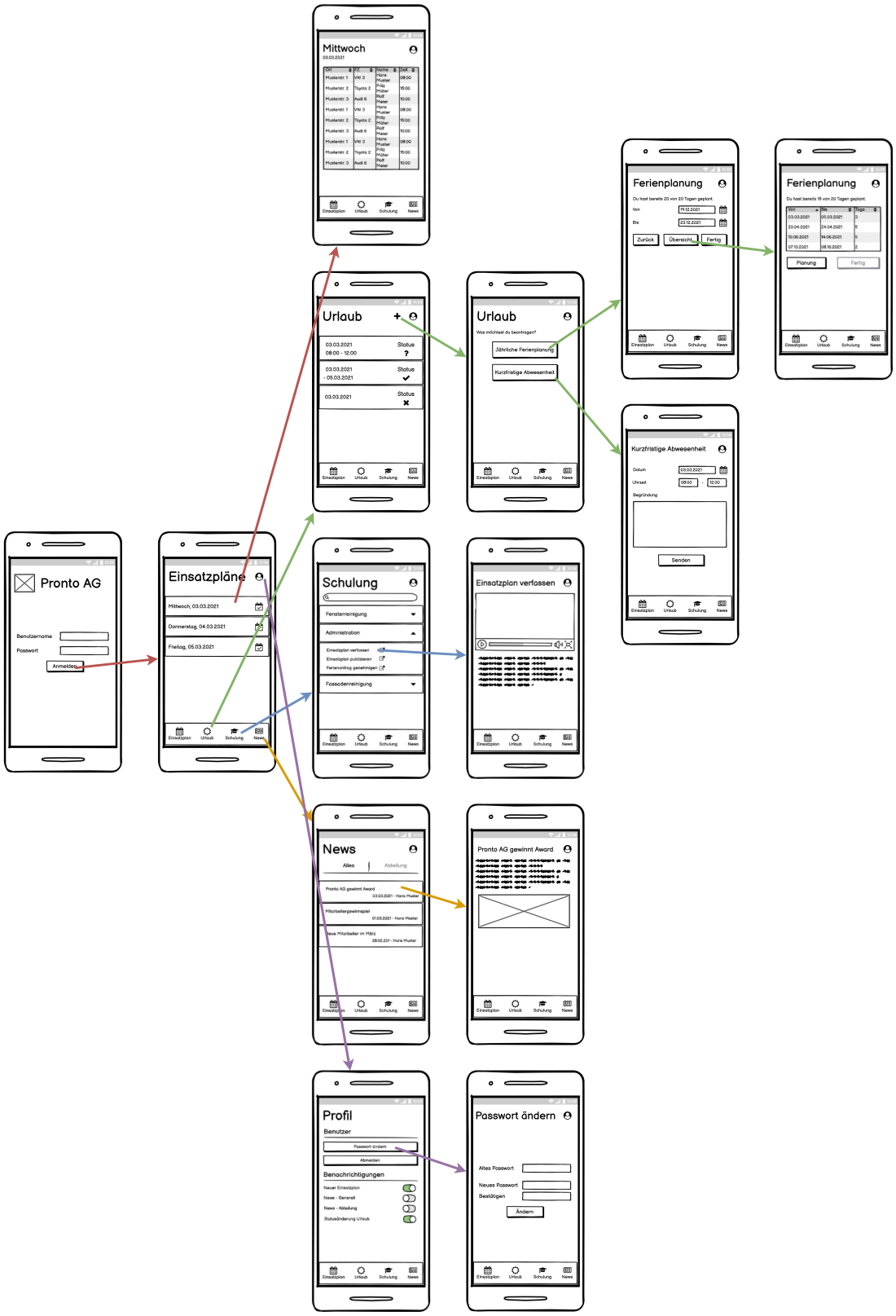

Abbildung 8.3: Übersicht Wireframes App

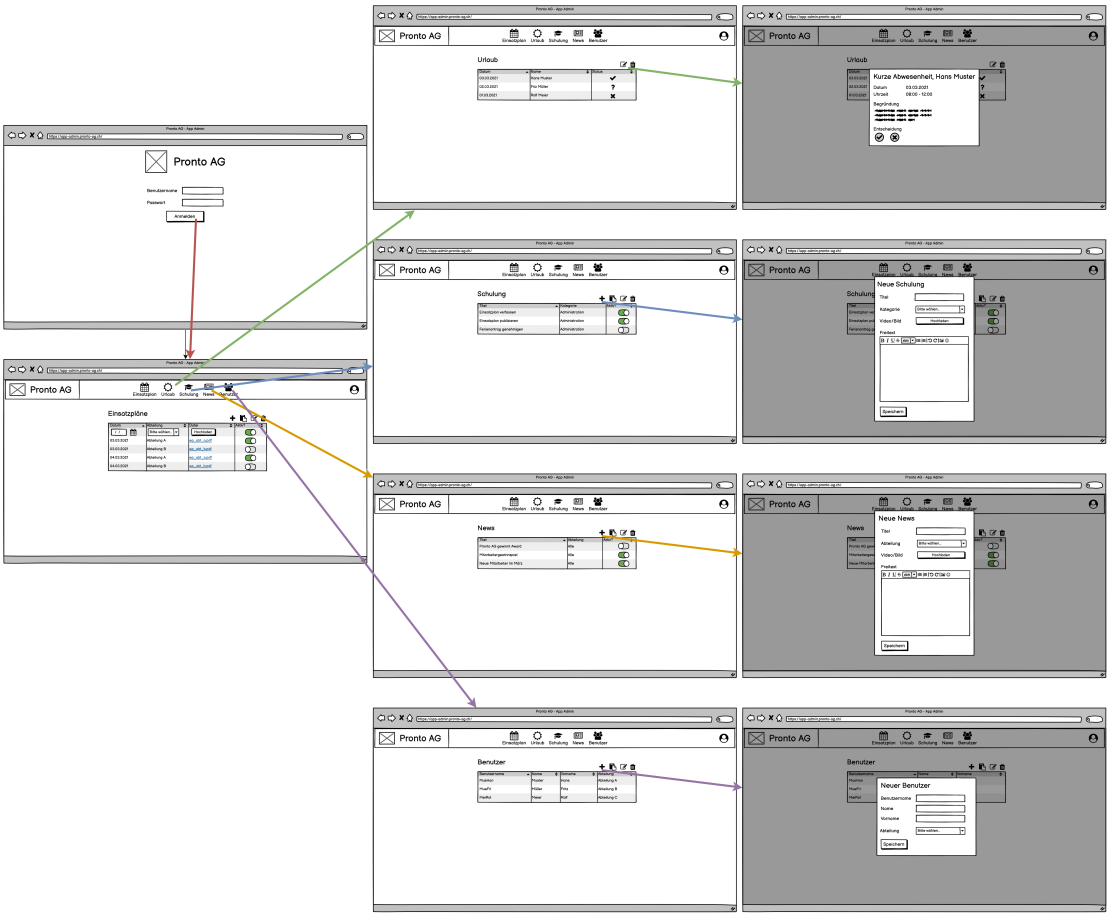

Abbildung 8.4: Übersicht Wireframes Administrations-Webseite

Eine Übersicht aller Wireframes ist unter Kapitel B.1.2 zu finden.

8.5 Releases

In diesem Abschnitt werden die erforderlichen und optionalen Funktionalitäten in Releases unterteilt. Die NFRs welche in diesem Projekt definiert wurden, müssen über die gesamte Laufzeit des Projektes beachtet werden und sind daher nicht in den Releases aufgeführt. Für die Versionierung wird sich an den Semantic Versioning 2.0 Standard gehalten.
Die Releases sind so aufgebaut, dass nach jedem Release ein fertiges Produkt verfügbar ist. Im Verlauf der Construction-Phase kann somit fortlaufend evaluiert werden, wie viele Releases noch in den Zeitplan passen. Da die Use Cases, siehe Kapitel 8.2, nach Priorität in die Releases aufgeteilt wurden und nicht jede Funktionalität zwingend eingebaut werden muss, können bei Zeitknappheit die höheren Releases nach Absprache mit der Betreuungsperson und dem Kunden weggelassen werden.

8.5.1 v0.1.0 – Prototyp

- Simple Autorisierung
- PDF von Server abrufen
- Push-Benachrichtigung wenn neues PDF auf dem Server verfügbar
- Lauffähig auf allen Plattformen (Android, iOS und Web)

8.5.2 v0.2.0 – Einsatzplan einfach

- US01.01 – Einsatzpläne verwalten (Ausbaustufe – Dateien)
- US01.02 – Einsatzpläne publizieren (Ausbaustufe – Firma + Benachrichtigung)
- US01.03 – Einsatzpläne abrufen (Ausbaustufe – Integriert)

8.5.3 v0.3.0 – Einsatzplan Abteilungsspezifisch

- US01.02 – Einsatzpläne publizieren (Ausbaustufe – Abteilungsspezifisch)
- US05.01 – Benutzer verwalten (Ausbaustufe – Abteilungsleiter)
- US05.01 – Benutzer verwalten (Ausbaustufe – Berechtigungen)
- US05.02 – Profil bearbeiten (Ausbaustufe – Passwort)

8.5.4 v0.4.0 – News einfach

- US04.01 – News abrufen (Ausbaustufe – Integriert + Firma)
- US04.02 – News verwalten (Ausbaustufe – Dateien + Firma)
- US04.03 – News publizieren (Ausbaustufe – Firma + Benachrichtigung)

8.5.5 v0.5.0 – News Hierachie

- US04.01 – News abrufen (Ausbaustufe – Hierarchie)
- US04.02 – News verwalten (Ausbaustufe – Hierarchie)
- US04.03 – News publizieren (Ausbaustufe – Hierarchie)
- US05.01 – Benutzer verwalten (Ausbaustufe – Hierarchie)

8.5.6 v0.6.0 – Schulungsvideos & Profil

- US03.01 – Schulungsvideos abrufen (Ausbaustufe – Integriert)
- US03.02 – Schulungsvideos verwalten (Ausbaustufe – Dateien)

8.5.7 v0.7.0 – Anleitungen

- US03.03 – Anleitungen abrufen (Ausbaustufe – Integriert)
- US03.04 – Anleitungen verwalten (Ausbaustufe – Dateien)

8.5.8 v0.8.0 – Urlaubsanträge

- US02.01 – Jahresurlaub erfassen (Ausbaustufe – Einfach)
- US02.02 – Jahresurlaub genehmigen (Ausbaustufe – Einfach)
- US02.03 – Freie Tage erfassen (Ausbaustufe – Einfach)
- US02.04 – Freie Tage genehmigen (Ausbaustufe – Einfach)

8.5.9 v0.9.0 – Urlaubsübersicht

- US02.05 – Persönliche Übersicht freie Tage einsehen (Ausbaustufe – Liste)
- US02.06 – Abteilungs-Übersicht freie Tage einsehen (Ausbaustufe – Liste)

8.5.10 v0.10.0 – Zusatz

- US01.03 – Einsatzpläne abrufen (Ausbaustufe – Kalender)
- US02.06 – Abteilungs-Übersicht freie Tage einsehen (Ausbaustufe – Kalender)
- US04.02 – News verwalten (Ausbaustufe – Integriert)

8.5.11 Changelog

25.05.2021	Da die Funktion zum Verwalten der Berechtigung von Benutzern bereits für v0.3.0 eingebaut werden konnte, wurde US05.01 (Ausbaustufe - Berechtigungen) von Version v0.5.0 nach Version v0.3.0 verschoben.
25.05.2021	Da die Funktion zum Ändern des eigenen Passwortes als sehr wichtig angeschaut wird, in dieser Arbeit die Applikation allerdings nur bis und mit v0.3.0 fertiggestellt werden kann, wurde entschieden US05.02 (Ausbaustufe – Passwort) von Version v0.6.0 nach Version v0.3.0 zu verschieben.

9 Framework Evaluation

Im Rahmen dieser Arbeit wird eine Applikation entwickelt, welche auf beiden im Moment gängigen Betriebssystemen für Mobiltelefone laufen muss, namentlich Android und iOS. Um eine solche Applikation umsetzen zu können, ohne für jedes Betriebssystem den Code neu schreiben zu müssen, ist ein cross-platform Framework notwendig. Dieses erlaubt es, einen Grossteil des Codes nur einmal zu schreiben und diesen dann auf mehreren Betriebssystemen nutzen zu können.

In diesem Kapitel werden diverse solcher Frameworks angeschaut und miteinander verglichen. Das Ziel war hierbei das am besten geeignete Framework, für die im Rahmen dieser Arbeit entwickelte Applikation, zu finden.

9.1 Einschränkung

Es gibt sehr viele Frameworks auf dem Markt, welche unsere Anforderungen potentiell erfüllen könnten. Aufgrund der begrenzten Zeit und des marginalen Nutzens wurden hier nicht alle uns bekannten Frameworks verglichen, sondern lediglich einige Wenige.

Bei der Auswahl der in diesem Kapitel aufgeführten Frameworks haben wir primär auf die Verbreitung und somit auf die Langlebigkeit und den Support geachtet. Je mehr ein Framework genutzt wird, desto wahrscheinlicher ist es, dass dieses noch lange bestehen wird und gute Antworten auf gängige Fragen vorhanden sind.

9.2 Anforderungen

Zu Beginn der Evaluation wurden die Anforderungen, welche an das zu verwendende Framework gestellt werden, definiert.

- Unterstützung für Push-Benachrichtigungen.
  - Siehe Begründung in Kapitel 6.1.3.
- Unterstützung für sowohl iOS wie auch Android als Betriebssystem.
- Ein Grossteil des Quellcodes kann zwischen den unterstützten Betriebssystemen geteilt werden.

9.3 Vergleich

Im Rahmen dieser Evaluation werden die folgenden Frameworks miteinander verglichen:

- Flutter
- React Native
- Xamarin
- Ionic

9.3.1 Flutter

Flutter ist ein von Google entwickeltes Framework für die cross-platform Entwicklung. Es ist für die Programmiersprache Dart verfügbar, welche ebenfalls von Google entwickelt wurde. Seit der Veröffentlichung erfreut es sich grosser Beliebtheit. Aufgrund dessen, wurde das Framework in diesen Vergleich einbezogen [4].

9.3.1.1 Erfüllung der Anforderungen

- Unterstützung für Push-Benachrichtigungen.
  - Das integrieren von Push-Benachrichtigungen wird mittels Firebase-Messaging-Service erreicht [5].
  - Die Integration kommt mit wenigen Betriebssystem-spezifischen Konfigurationen aus und erlaubt es einen Grossteil des Benötigten Codes zentral zu definieren [5].
- Unterstützung für sowohl iOS wie auch Android als Betriebssystem.
  - Beide Betriebssysteme werden von Flutter unterstützt [4].
- Ein Grossteil des Quellcodes kann zwischen den unterstützten Betriebssystemen geteilt werden.
  - Flutter wandelt den Sourcecode, welcher in Dart geschrieben wird, in nativen Code für die jeweilige Plattform um. Entsprechend kann ein Grossteil des Quellcodes zwischen den Plattformen geteilt werden [6].

9.3.1.2 Zusätzliches

Flutter bietet Hot-Reloading an, dies bedeutet, dass die Ergebnisse von Codeänderungen direkt auf dem Entwicklungsgerät sichtbar werden. Auch bietet Flutter die Kompilierung der App in eine Webseite an. Dieses Feature ist seit Flutter 2.0 unterstützt [7]. Zusätzlich ist auch die Desktop-Integration für MacOS, Windows und Linux unterstützt. Diese Funktionalität befindet sich allerdings noch in der Beta-Phase.

9.3.1.3 Fazit

Flutter ist ein Framework mit vielen guten Konzepten. Die Sprache Dart scheint intuitiv zu sein. Das Verwenden der selben Benutzeroberfläche für Android und iOS kann für einige Nutzer verwirrend sein, welche sich den Stil des jeweiligen Betriebssystems gewohnt sind.

Positiv:

- Angebotene Kompilierung für das Web.
- Einfache Integration von Push-Benachrichtigungen.
- Keine Altlasten, da Dart erst seit 2015 und Flutter seit 2018 existiert.

Negativ:

- Eine neue Programmiersprache müsste erlernt werden (Dart).

9.3.2 React Native

React Native ist eine auf dem React-Framework basierende Lösung für die cross-platform Entwicklung. Während React selbst in Web-Applikationen Verwendung findet, wird React Native für die Entwicklung mobiler Applikationen auf iOS und Android verwendet. Beide Frameworks wurden von Facebook entwickelt [8].

9.3.2.1 Erfüllung der Anforderungen

- Unterstützung für Push-Benachrichtigungen.
  - Die Integration von Push-Benachrichtigungen kann über verschiedene Drittanbieter Tools gemacht werden. Von React Native selbst wird Expo (`https://expo.io`) empfohlen. Expo ist ein Set von Tools, welches die Handhabung von React Native vereinfacht [9]. Dieses setzt sowohl Firebase als auch den Apple-Push-Dienst ein, um die Nachrichten zu verschicken. Alternativ kann auch direkt Firebase oder ein anderer Drittanbieter eingebunden werden.
- Unterstützung für sowohl iOS wie auch Android als Betriebssystem.
  - Beide Betriebssysteme werden von React Native unterstützt [8].
- Ein Grossteil des Quellcodes kann zwischen den unterstützten Betriebssystemen geteilt werden.
  - Grundsätzlich kann der gleiche Code für alle unterstützten Systeme eingesetzt werden. Sobald allerdings spezifischere Funktionen notwendig sind, so muss dies plattformspezifisch abgewickelt werden.

9.3.2.2 Zusätzliches

Mithilfe von react-native-web oder den Expo-Tools kann eine React Native Applikation auch im Web laufen. Bei Verwendung der Expo-Tools ist kein Android Studio oder Xcode auf dem Entwicklergerät notwendig, da Cloud-Builds durchgeführt werden können. Die Expo-Builds setzen allerdings einen Account bei Expo voraus.

9.3.2.3 Fazit

React Native ist ein Framework, welches auf dem viel genutzten React aufbaut. Die verwendete Programmiersprache und die Konzepte sind bereits bekannt und erprobt. Für Funktionen wie Push-Benachrichtigungen sind Drittanbieter wie Expo oder Firebase notwendig, um nicht für jede Plattform das native Messaging einsetzten zu müssen.

Positiv:

- Mittels Expo ist Cloud-Kompilierung möglich.
- Funktioniert auch für das Web.
- Nutzt JavaScript zur Programmierung. Diese Programmiersprache ist dem Entwicklerteam schon bekannt.

Negativ:

- Limitierte Unterstützung ohne Drittanbieter.
- Keine einfache Implementation von Push-Benachrichtigungen.

9.3.3 Xamarin

Xamarin ist ein von Microsoft entwickeltes cross-platform-Framework, welches die Programmiersprache C# nutzt [10].

9.3.3.1 Erfüllung der Anforderungen

- Unterstützung für Push-Benachrichtigungen.
  - Die Integration von Push-Benachrichtigungen kann in den Plattformspezifischen Projekten mithilfe der jeweiligen Anbieter (Android - Firebase, iOS - Apple-Push-Service) geschehen.
  - Alternativ kann bei Verwendung von Xamarin.Forms der Azure Notification Hub verwendet werden. Dieser vereint die beiden oben genannten Services miteinander [11].
- Unterstützung für sowohl iOS wie auch Android als Betriebssystem.
  - Beide Betriebssysteme werden von Xamarin unterstützt [10].
- Ein Grossteil des Quellcodes kann zwischen den unterstützten Betriebssystemen geteilt werden.
  - Grundsätzlich kann in Xamarin 75% des Codes geteilt werden. Hierbei muss lediglich das UI plattformspezifisch designt werden. Falls Xamarin.Forms verwendet wird, kann sogar über 90% des Codes geteilt werden, da das UI dann auch plattformübergreifend definiert werden kann.
  - Xamarin.Forms ist ein Projekt von Xamarin, in welchem vordefinierte UI-Komponenten für die unterstützten Plattformen zur Verfügung stehen. Entsprechend steht allerdings nicht mehr der volle Funktionsumfang der Plattformen zur Verfügung.

9.3.3.2 Zusätzliches

Xamarin bietet zusätzlich die Möglichkeit Desktop-Applikationen für Windows zu erstellen. Dies mithilfe von UWP oder WinUI [12].

9.3.3.3 Fazit

Xamarin ist ein gut durchdachtes Framework mit vielen Funktionalitäten. Es bindet sich gut in bestehende Windows-Umgebungen ein. Da es von Microsoft vertrieben wird, ist die Unterstützung von Drittanbieter-Produkten in der Entwicklung, sowie im Deployment beschränkt.

Positiv:

- Unterstützt Windows-Desktop mit UWP und WinUI.
- Programmiersprache ist C#. Diese ist dem Entwicklerteam schon bekannt.
- Viel geteilter Code mit Xamarin.Forms.

Negativ:

- Implementierung von Push-Benachrichtigungen mithilfe von Azure. Dies fügt neben Firebase und dem Apple-Push-Service eine neue Abhängigkeit zur Implementation hinzu.
- Keine einfache Implementation von Push-Benachrichtigungen.
- Xamarin.Forms wird voraussichtlich 2022 von .NET Multi-platform App UI (MAUI) abgelöst. Dieses wird allerdings erst mit .NET 6 ausgeliefert werden, was momentan nur als Preview verfügbar ist [13].
- Die Entwicklung auf Linux ist von Xamarin nicht unterstützt.

9.3.4 Ionic

Ionic ist ein, von der gleichnamigen Firma entwickeltes, Framework welches auf Javascript basiert. Ionic kann mit den bekannten JavaScript Frameworks Vue, React und Angular kombiniert werden um dem Entwickler seine bekannte Umgebung zu gewährleisten [14].

9.3.4.1 Erfüllung der Anforderungen

- Unterstützung für Push-Benachrichtigungen.
  - Die Integration von Push-Benachrichtigungen kann mithilfe des Cordova-Plugins „cordova-plugin-push“ realisiert werden. Dieses benutzt wiederum Firebase um die Benachrichtigungen auf die Endgeräte zu verteilen [15].
  - Die Dokumentation von Ionic verweist noch auf „phonegap-plugin-push “ als Plugin, welches aber veraltet ist. Die Verwendung des Nachfolgers ist bei Ionic nicht dokumentiert [15].
- Unterstützung für sowohl iOS, wie auch Android als Betriebssystem.
  - Beide Betriebssysteme werden von Ionic unterstützt [14].
- Ein Grossteil des Quellcodes kann zwischen den unterstützten Betriebssystemen geteilt werden.
  - Der Code kann geteilt werden. Konfigurationen wie Berechtigungen und Icons müssen allerdings plattformspezifisch definiert werden.

9.3.4.2 Zusätzliches

Speziell zu erwähnen ist, dass die Komponenten von Ionic auf das jeweilige Layout der Betriebssysteme angepasst sind. Dies bedeutet, dass ein Ionic-Knopf auf Android anders aussieht als auf iOS. Dadurch finden sich die Benutzer, auf ihrer gewohnten Plattform, in der Applikation besser zurecht [16].

9.3.4.3 Fazit

Ionic ist ein Framework, welches mit allen gängigen JavaScript-Frameworks zusammenarbeiten kann. Viele Komponenten sind bereits vorkonfiguriert und passen sich dem Stil des jeweiligen Betriebssystems an. Die Performance ist allerdings nicht optimal und es hängt viel Funktionalität von Plugins ab.

Positiv:

- Nutzt JavaScript und bekannte Frameworks.
- Viele vordefinierte UI-Komponenten.
- Viele verfügbare Plugins.

Negativ:

- Implementierung von Push-Benachrichtigungen komplex.
- Veraltete Dokumentation zum Thema Push-Benachrichtigungen [15].
- Nicht sehr performant.
- Viele Abhängigkeiten zu Plugins.

9.4 Entscheidung

Aufgrund des vorgenommenen Vergleichs aus Kapitel 9.3 hat sich das Entwicklerteam für die Verwendung des, in Kapitel 9.3.1 evaluierten, Flutter-Framework entschieden. Von den evaluierten Frameworks waren schlussendlich noch Flutter und Xamarin in der engeren Auswahl. Hierbei konnte Xamarin vor allem durch die Verwendung von C# als Programmiersprache punkten. Die Entscheidung für Flutter statt Xamarin ist auf die nachfolgenden Vorteile zurückzuführen:

- Die Einrichtung der Entwicklungsumgebung ist bei Flutter sehr intuitiv und einfach.
- Flutter erlaubt die Entwicklung auf allen Betriebssystemen mit einer selbst gewählten IDE.
- Push-Benachrichtigungen können in Flutter sehr einfach und unkompliziert integriert werden.

10 API Evaluation

Die im Rahmen dieser Arbeit entwickelte Applikation stellt einige Anforderungen bezüglich der zentralen Speicherung und Abrufbarkeit von Daten. Aufgrund dessen wird eine Serverapplikation geschrieben werden, welche eine API zur Verwaltung eben dieser Daten anbietet.

In diesem Kapitel werden verschiedene Techniken, welche die Implementation einer solchen API ermöglichen begutachtet und miteinander verglichen. Mithilfe dieser Analyse soll die für die Applikation am besten geeignete Technologie gefunden werden.

10.1 Einschränkung

Auf dem Markt gibt es diverse Möglichkeiten wie APIs aufgebaut werden können. Um diese Evaluation in einem nützlichen Rahmen zu halten, wurden bereits im Vorfeld drei mögliche Technologien ausgewählt, welche hier genauer analysiert werden sollen.

Bei der Auswahl dieser Kandidaten wurde primär auf deren Verbreitung im Markt geachtet.

10.2 Anforderungen

Im Gegensatz zum Framework bei welchem wir bereits vor der Evaluation klare Anforderungen definieren konnten, gibt es bei der API keine hart definierten Kriterien. Es gibt allerdings einige Punkte, auf welche bei der Evaluation geachtet wird. Diese sind nachfolgend aufgelistet:

- Einfache Einbindung einer Authentisierung.
- Einfach zu Dokumentieren.

10.3 Vergleich

Im Rahmen dieser Evaluation werden die folgenden API-Technologien miteinander verglichen:

- REST
- GraphQL
- gRPC

10.3.1 REST

Die klassische und traditionelle Variante wie APIs definiert werden, geschieht mithilfe von REST. REST steht hierbei für „Representational State Transfer“. Bei dieser Art von API werden Ressourcen definiert und Vorgänge mithilfe von Verlinkungen abgebildet, um die auf die API zugreifende Software durch den Lebenszyklus einer Ressource durchzuführen [17]. REST-APIs setzten auf HTTP auf und benutzen auch die in HTTP festgelegten Methoden.

Positiv:

- Trennung von Server und Client.
- Gute Übersichtlichkeit mit definierten Ressourcen.
- Gut skalierbar, da jeder Aufruf komplett unabhängig ist.
- Prozessabläufe können mithilfe von Verlinkungen abgebildet werden.

Negativ:

- Die API muss separat dokumentiert werden.
- Der Client muss immer die gesamte Ressource abrufen, was Bandbreite kosten kann.
- Um eine komplett REST-Konforme API zu implementieren, müssen viele Faktoren beachtet und Verlinkungen gemäss HATEOAS eingebaut werden. Einige dieser Funktionalitäten sind im Rahmen dieser Arbeit nicht notwendig und würden zusätzlichen Mehraufwand bedeuten.

10.3.1.1 Fazit

REST ist eine etablierte und die wahrscheinlich meistgenutzte API Technik welche im Moment existiert. Die Vorgaben sind strukturiert und gut durchdacht. Eine vollkommen konforme REST API zu erstellen, bedeutet allerdings viele Vorgaben und Anforderungen für den Entwickler [17].

10.3.2 GraphQL

GraphQL ist eine Abfragesprache für APIs welche spezifisch dafür konzipiert wurde, dem Client genau die Informationen zu liefern welche dieser auch benötigt. Die Technik erlaubt es, verschachtelte und komplexe Anfragen zu definieren [18].

Positiv:

- Das in der Sprache integrierte Schema ermöglicht die automatische Dokumentation der API. Dies setzt allerdings voraus, dass die Definition genügend Informationen enthält um selbsterklärend zu sein.
- GraphQL erlaubt Variablen und komplexe Abfragen welche eine gute Filterung der Daten vom Client aus ermöglichen.
- Die API kann ohne Versionierung um neue Felder erweitert werden, da der Client nur die von ihm erforderlichen Felder abfragt.

Negativ:

- Fehler werden nicht mittels HTTP-Statuscode signalisiert sondern mit dem Inhalt der GraphQL-Antwort.
- Im Gegensatz zu REST, welches fixe Endpunkte hat, kann GraphQL nicht von HTTP-Caching profitieren auch wenn eine Abfrage eigentlich bereits einmal so abgesetzt wurde.
- GraphQL ist von sich aus relativ Komplex. Simple APIs können in REST daher schneller implementiert werden.

10.3.2.1 Fazit

GraphQL wurde entwickelt um mit sich schnell ändernden API-Anforderungen und Daten umzugehen. Es wird allerdings auch das Problem des Overfetching sowie Underfetching gelöst. Ausserdem stellt GraphQL dem Client mächtige Tools zur Filterung der von ihm erhaltenen Daten zur Verfügung. GraphQL macht hierbei allerdings nicht von den in HTTP integrierten Funktionalitäten Gebrauch [19] [20].

10.3.3 gRPC

gRPC ist eine API Technik welche primär auf Geschwindigkeit und Datensparsamkeit ausgerichtet ist. Ausserdem unterstützt gRPC eine Vielzahl von Plattformen und Betriebssystemen [21].

Positiv:

- gRPC ist sehr performant und kann Daten in hoher Geschwindigkeit übertragen.
- Eingebaute Authentisierung.
- Braucht in der Standardkonfiguration weniger Daten zum Übertragen der gleichen Information wie REST und GraphQL.
- Viel Code wird von gRPC selbst generiert um den Service, welcher vom Entwickler entworfen wurde, zu implementieren.

Negativ:

- Limitierte Unterstützung für den Browser. gRPC-Web schafft hier allerdings Abhilfe mit eingeschränktem Umfang.
- Die übertragenen Daten sind im Binärformat und somit für Menschen nicht lesbar. Dies kann bei der Problemfindung hinderlich sein.
- Der Client muss die gRPC-Spezifikation des Servers besitzen, um Nachrichten mit diesem austauschen zu können.

10.3.3.1 Fazit

gRPC wurde für skalierbare und grosse Ökosysteme entwickelt, in welchen Geschwindigkeit und Bandbreite eine grosse Rolle spielen. Durch die breite Unterstützung diverser Sprachen und System kann gRPC auf fast jedem Gerät eingesetzt werden. Die Unterstützung für das Web wurde leider erst spät angegangen, ist nun allerdings auch implementiert [22].

10.4 Entscheidung

Aufgrund des vorgenommenen Vergleichs aus Kapitel 10.3 hat sich das Entwicklerteam für die Verwendung der, in Kapitel 10.3.2 evaluierten, GraphQL Technologie entschieden. Dies aus den folgenden Gründen:

- GraphQL verhindert effizient Overfetching und Underfetching und erlaubt es so nur die gewollten Daten abzurufen.
- GraphQL erleichtert mit seinem, in der Technologie eingebauten, Schema das erstellen von Dokumentation enorm.
- In GraphQL kann die Authentisierung mithilfe von den Entwicklern bekannten und bewährten Technologien wie JWT-Token implementiert werden.

Teil III

Architektur

11 | Übersicht

Im Teil „Architektur“ werden zum Einen die Entwicklungstools evaluiert und dokumentiert, welche für die Umsetzung des Projekts benötigt werden. Zum Anderen wird in diesem Teil auch die gesamte Architektur des Projekts beschrieben.

Die zugrundeliegenden Entscheidungen, für die Wahl von Technologien und Architekturprinzipien, werden unter Berücksichtigung der Evaluationen aus der Vorstudie und in Abstimmung mit der Implementation des technischen Prototyps aus diesem Kapitel getroffen. Allen Entscheidungen liegt hierbei der Grundsatz des Single-Responsibility-Prinzip zugrunde, welches das Projekt stark beeinflusst hat. Sollten andere Faktoren bei Entscheidungen eine wichtige Rolle gespielt haben, so ist dies dokumentiert. Zusätzlich werden einige Entscheidungen im Abschnitt Herausforderungen in Kapitel 18 weiter ausgeführt.

12 Systemübersicht

In diesem Kapitel wird die geplante Architektur in Form eines Deployment-Diagramm dargestellt. Ausserdem werden die einzelnen Komponenten des Diagramms textuell erläutert.

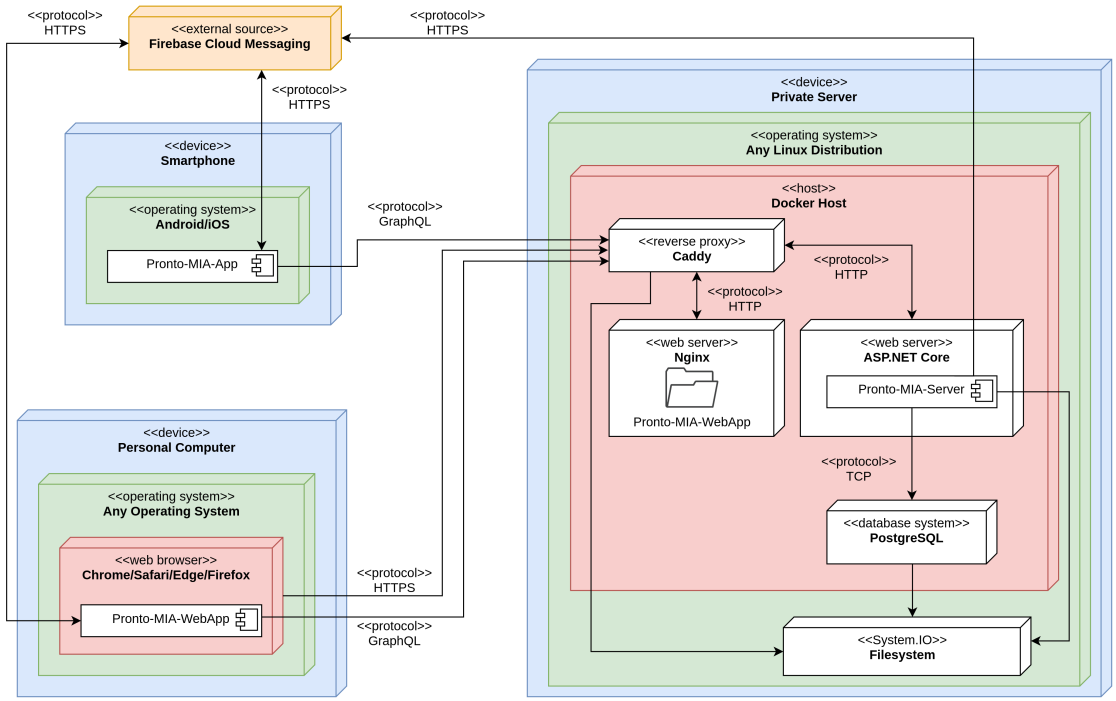

Abbildung 12.1: Deploymentdiagramm des Endprodukts

Komponente	Beschreibung
Pronto-MIA-App	Stellt das User-Interface auf iOS und Android zur Verfügung. Kommuniziert mit dem Server.
Pronto-MIA-WebApp	Stellt das User-Interface im Browser zur Verfügung. Kommuniziert mit dem Server.
Caddy	Leitet eingehende Anfragen an die Web-App oder die API weiter. Verwaltet SSL-Zertifikate.
Nginx	Liefert die Pronto-MIA-WebApp aus.
Pronto-MIA-Server	Stellt die API für das Frontend zur Verfügung. Sendet Push-Benachrichtigungen an die Firebase API um diese an die Geräte zu verteilen.
PostgreSQL	Persistiert die Daten der API.
Filesystem	Die Konfigurationen und Dateien von verschiedenen Containern sind hier abgelegt. **Caddy:** Die Konfiguration von Caddy sowie die gespeicherten SSL-Zertifikate. **Pronto-MIA-Server:** Die Konfiguration sowie hochgeladenen PDF-Dateien. **PostgreSQL:** Die persistierten Daten der Datenbank.
Firebase Cloud Messaging	Sendet Push-Benachrichtigungen zu den verschiedenen Geräten, auf welchen das Frontend läuft. Nimmt Nachrichten vom Server entgegen und verteilt diese.

13 Technologien

In diesem Kapitel werden die verwendeten Technologien und Softwares, welche zur Umsetzung der Applikation benötigt werden, dokumentiert und näher erläutert. Bei jeder verwendeten Software ist angegeben, zu welchem Zweck diese verwendet wird.

13.1 Frontend

Im Rahmen der Vorstudie wurde, während der Framework Evaluation aus Kapitel 9, eine geeignete cross-platform Technologie gesucht. Die Wahl fiel hierbei auf Flutter. Flutter bietet, wie viele heutzutage gängige Frontend Frameworks, die Möglichkeit, komponentenbasierte Benutzeroberflächen in Form von Widgets zu implementieren. Da dies aber in einer vollständigen Frontend-Architektur nur den Presentation-Layer abdeckt und Flutter keine Architektur vorgibt, sind weitere Libraries nötig um eine saubere Struktur erreichen zu können. Während der Implementation des Prototyps, haben sich die folgenden weiteren Anforderungen an die Frontend-Architektur herauskristallisiert:

- State Management
- Navigation
- Service Management

Flutter bietet grundsätzlich mit `StatefulWidget` eine Möglichkeit an, State Management zu betreiben. Hierbei gibt es aber einige Herausforderungen zu bewältigen:

- Dadurch, dass `StatefulWidget` sich konzeptionell nicht von anderen Widgettypen unterscheidet, ist es schwierig eine saubere Trennung von Benutzeroberfläche und Business-Logik zu realisieren.
- Die Weitervererbung des State an darunterliegende Widgets muss auf jeder Stufe manuell gemacht werden. `StatefulWidget` bietet keine Möglichkeit an, den State beliebig tief weiterzuvererben. `InheritedWidget` ist Flutters Ansatz diese Herausforderung zu bewältigen, jedoch wird dieser innerhalb der Community als unintuitiv angesehen. Sogar in der offiziellen Dokumentation zum simplen Flutter State Management wird unter anderem `InheritedWidget` als „low-level“ bezeichnet und man wird auf eine Library verwiesen [23].

Die Navigation zwischen verschiedenen Ansichten innerhalb der Applikation wird in Flutter über die Navigator API abgewickelt [24]. Problematisch bei diesem Vorgehen ist, dass beim Navigieren mit dem Navigator immer der sogenannte `BuildContext` mitgegeben werden muss. Der `BuildContext` wird grundsätzlich vom Framework selbst instanziert und der Applikation in den build-Methoden der Widgets, welche zuständig für den Aufbau der Benutzeroberfläche sind, übergeben. Möchte man über eine Methode, die sich in einer beliebigen Komponente der Applikation befindet, auf ein neues Widget navigieren, muss man darauf achten, dieser Methode den `BuildContext` zu übergeben. Dies kann beispielsweise dazu führen, dass ein Klick-Event auf einem Button, welches Zugriff auf den `BuildContext` hat, diesen über mehrere Methoden und Klassen hinweg bis zur eigentlichen Navigation übergeben muss.

Eine letzte wichtige Anforderung die sich aufgezeigt hat ist Service Management. Flutter bietet nativ keine Möglichkeit an um Services, wie Singletons und Factories, zu verwalten. Grundsätzlich wäre es möglich eine solche Service Management Komponente selbst zu implementieren, jedoch ist dies mit beträchtlichem Zeitaufwand verbunden.

Verwendete Software

- Flutter (`https://flutter.dev/`)
  - Ging in der Framework Evaluation als Sieger hervor.
  - Fungiert als Grundlage für das Frontend des Projekts.
- Stacked (`https://pub.dev/packages/stacked`)
  - Stellt eine MVVM-basierte Architektur zur Verfügung, welche die oben genannten Herausforderungen abdeckt.
  - Ermöglicht es ViewModels zu implementieren, welche zur sauberen Trennung von Benutzeroberfläche und Logik in der Applikation dienen.
  - Bietet einen Navigationsservice an, welcher unabhängig vom `BuildContext` verwendet werden kann.
  - Bietet einen Service Locator fürs Management von Dependencies, wie Singletons und Factories an.
- GetIt (`https://pub.dev/packages/get_it`)
  - Implementiert den bereits erwähnten Service Locator.
  - Wird als Teil des Stacked Pakets ausgeliefert.
- GetX (`https://pub.dev/packages/get`)
  - Implementiert State Management.
  - Ermöglicht die Navigation unabhängig vom `BuildContext`.
  - Wird als Teil des Stacked Pakets ausgeliefert.

13.1.1 Schnittstelle zum Server

In der API Evaluation, aus Kapitel 10, wurde entschieden GraphQL als zentrale Schnittstelle zwischen Client und Server zu verwenden. Bei GraphQL Libraries für Flutter wird grundsätzlich zwischen zwei verschiedenen Varianten unterschieden:

1. Bei der ersten Variante handelt es sich um herkömmliche GraphQL Clients, welche grundsätzlich Methoden zum Versenden von Queries und Mutationen anbieten. Diese sind oftmals nicht direkt an Flutter als Framework gebunden und finden daher auch in anderen Dart Projekten Verwendung. Ein Beispiel für eine solche Library ist „GraphQL Client“ (https://pub.dev/packages/graphql).
2. Bei der zweiten Variante handelt es sich um Libraries die Wrapper-Widgets anbieten um die GraphQL Queries und Mutationen direkt an die Benutzeroberfläche zu binden. Diese bauen meist auf einer Library der ersten Variante auf. Ein Beispiel für eine solche Library ist „GraphQL Flutter“ (https://pub.dev/packages/graphql_flutter).

Aufgrund dessen, dass bei der zweiten Variante die GraphQL Queries direkt an Widgets angebunden werden, wird in diesem Projekt die erste Variante vorgezogen. Dadurch kann eine flexiblere und von Flutter unabhängigere Architektur aufgebaut werden.

Eine zusätzliche Anforderung an einen solchen GraphQL Client ist die Möglichkeit neben herkömmlichen Queries auch das Hochladen von Dateien auf den Server zu ermöglichen. Dafür muss der Client die „GraphQL Multipart Request Specification“ implementieren [25]. Dies wird beispielsweise in UC01 benötigt, um Einsatzpläne hochladen zu können.

Verwendete Software

- GraphQL Client (https://pub.dev/packages/graphql)
  - Stellt einen Flutter-unabhängigen GraphQL Client zur Verfügung mit Support für Queries, Mutations und Subscriptions.
  - Ermöglicht das Hochladen von Dateien mittels sogenannter Multipart Requests.
  - Implementiert eine Cache-Lösung für die Zwischenspeicherung von Queries.

13.1.2 Anzeige von PDFs

UC01, UC03 und UC04 beinhalten alle die Anforderung PDF-Dateien innerhalb der Applikation abzurufen. Flutter bietet einige Libraries an, welche es ermöglichen PDF-Dateien innerhalb des Apps darzustellen. Beim Testen einiger dieser Libraries ist jeweils mindestens eines der folgenden Probleme zum Vorschein gekommen:

- Die Library ist nicht für alle nötigen Plattformen geeignet. Meist fehlt der Support für Web.
- Die Library hängt von älteren Versionen anderer Libraries ab. Aufgrund dessen, dass Dependencies in Flutter jeweils nur in einer spezifischen Version installiert werden können, führt dies zu Versionsproblemen. Hierbei muss dann, zur Lösung des Konfliktes, auf die ältere Version der entsprechenden Library zurückgegriffen werden.¨
- Die Navigation innerhalb des PDFs ist auf mindestens einer Platform nicht ideal. Bei der Library PDF Render (https://pub.dev/packages/pdf_render) ist es beispielsweise nicht möglich mit der Maus durch das PDF zu scrollen, da sie ursprünglich für mobile Geräte mit Touchscreen konzipiert wurde.

Aufgrund dieser Umstände wurde schlussendlich entschieden, die Darstellung von PDF-Dateien platformabhängig zu implementieren. Für mobile Geräte soll die Darstellung über einen, im App integrierten, Viewer abgewickelt werden, während bei der Desktopversion über Web das PDF in einem neuen Browsertab geöffnet werden soll.

Verwendete Software

- Flutter PdfView (https://pub.dev/packages/flutter_pdfview)
  - Ermöglicht die Darstellung von PDF-Dateien innerhalb des Apps für Android und iOS
- URL Launcher(https://pub.dev/packages/url_launcher)
  - Ermöglicht das Öffnen einer heruntergeladenen PDF-Datei in einem neuen Browsertab.

13.2 Backend

Im Rahmen der API Evaluation aus Kapitel 10 wurde GraphQL als Technologie gewählt. Nachdem die API-Art bestimmt war, musste nun noch eine geeignete Programmiersprache und API-Library gefunden werden. Da im Unterricht bereits mit der GraphQL-Library „HotChocolate“ gearbeitet wurde und diese somit bekannt war, fiel die Wahl auf diese [26]. Dadurch war auch die zu verwendende Programmiersprache, namentlich C#, gegeben.

Obschon „HotChocolate“ es theoretisch erlaubt, GraphQL-Queries über alle möglichen Medien zu übertragen, muss die Frontend-Applikation eine einfache Möglichkeit haben mit der API zu kommunizieren. Hier bietet sich HTTP als Protokoll an. Da „HotChocolate“ eine gute Integration mit dem von Microsoft entwickelten Web-Framework „ASP.NET“ bietet, wurde dieses als Bindeglied zwischen der API und der Frontend-Applikation gewählt. Dadurch kann auch von den vielen Funktionalitäten von „ASP.NET“ profitiert werden. Service-Management sowie Dependency Injection, wie auch Authentisierung und Autorisierung, sind im Framework bereits verbaut und müssen nur noch korrekt implementiert werden.

Ein weiterer zu beachtender Aspekt war die Persistenz. Die Daten der Applikation mussten persistiert werden, um diese auch über Neustarts hinweg abrufen zu können. Hier bot sich klassisch eine Datenbank an. Da C# als Programmiersprache verwendet wird und keine SQL-Queries direkt geschrieben werden sollten, musste ein ORM-Mapper bestimmt werden, welcher objektorientierte Strukturen in eine relationale Datenbank abspeichern kann. Der bekannteste und auch meistgenutzte ORM-Mapper für C# ist das „Entity Framework“. Da mit diesem bereits die meisten Erfahrungen im Team vorhanden waren, wurde dieser gewählt.

Verwendete Software

- HotChocolate (https://github.com/ChilliCream/hotchocolate)
  - Im Team bekannte Implementation von GraphQL
  - Fungiert als Grundlage für die API
- ASP.NET core (https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0)
  - Ist ein cross-platform Web-Framework, entwickelt von Microsoft
  - Kann gut mit „HotChocolate“ kombiniert werden.
  - Bietet Funktionalitäten wie Authentisierung, Autorisierung und Service Management, sowie Dependency Injection
  - Verbindet die verschiedenen Komponenten des Backend miteinander
- Entity Framework (https://docs.microsoft.com/en-us/ef/)
  - Ist ein ORM-Mapper für C# von Microsoft
  - Besitzt eine gute Integration mit ASP.NET
  - Bietet Funktionalitäten wie Lazy Loading und Transaktionen

13.2.1 Datenbank

Nachdem der ORM-Mapper bestimmt war, musste definiert werden, welche Datenbank im Hintergrund für die Persistenz zuständig ist. Aufgrund des grossen Erfahrungsschatzes des Teams mit „PostgreSQL“, wurde diese Datenbank ausgewählt um die Daten zu persistieren.

Verwendete Software

- PostgreSQL (https://www.postgresql.org/)
  - Dem Team bekannte relationale Datenbank
  - Unterstützt Transaktionen
  - Besitzt eine gute Integration mit „Entity Framework“

13.2.2 Speicher von Dateien

Um die in der Frontend-Applikation hochgeladenen Dateien zu persistieren, musste eine Lösung gefunden werden. Da das Speichern grosser Dateien nicht eine Hauptaufgabe von relationalen Datenbanken dieser Art darstellt und diese entsprechend nicht dafür optimiert sind, ist eine Speicherung solcher in der Datenbank nicht zu empfehlen. Entsprechend musste eine Lösung gefunden werden, wie die Dateien persistiert werden können. Um dieses Problem zu lösen, kann die in C# integrierte „System.IO“-API verwendet werden. Diese erlaubt der Applikation den Zugriff auf das Dateisystem des Servers und somit die Ablage von Dateien.

Verwendete Software

- System.IO-API (https://docs.microsoft.com/en-us/dotnet/api/system.io?view=net-5.0)
  - In C# integrierte API
  - Bietet Funktionalität zur Verwaltung von Ordnern und Dateien
- System.IO.Abstractions (https://github.com/System-IO-Abstractions/System.IO.Abstractions)
  - Abstrahiert die System.IO-API hinter einem Interface
  - Ermöglicht es System.IO-Operationen testen zu können

13.3 Allgemein

In diesem Kapitel sind Anforderungen gelistet, welche von sowohl Backend als auch Frontend erfüllt werden mussten. Entsprechend ist bei jeder Anforderung die eingesetzte Software in Front- und Backend aufgeteilt.

13.3.1 Push-Benachrichtigungen

Um eine zeitnahe Information aller betroffenen Mitarbeitenden der Pronto AG zu gewährleisten, sollen innerhalb der Applikation Push-Benachrichtigungen verwendet werden. Diese müssen sowohl im Frontend, wie auch im Backend implementiert werden. Im Frontend um die Nachrichten zu empfangen und im Backend um diese zu senden.

Um Push-Benachrichtigungen verwenden zu können, wird auf das Framework namens “Firebase„ von Google gesetzt. Dieses hat eine einfache Integration mit Flutter und kann alle angebotenen Zielplattformen abdecken. Aus dem grossen Baukasten, welchen “Firebase„ bieten würde, wird allerdings lediglich die Komponente “Firebase Messaging„ verwendet, um Push-Benachrichtigungen senden zu können.

Verwendete Software

- Frontend
  - Firebase Core (https://pub.dev/packages/firebase_core)
    - Ermöglicht die Verbindung zwischen der Flutter App und Firebase Service
  - Firebase Messaging (https://pub.dev/packages/firebase_messaging)
    - Stellt Funktionalitäten von Firebase Messaging für Flutter zur Verfügung
    - Wird benötigt um Geräte zu (de)registrieren und Nachrichten zu empfangen
- Backend
  - Firebase Admin (https://firebase.google.com/docs/reference/admin)
    - Angeboten von Google

    - Stellt eine Programmierschnittstelle für C# zur Verfügung über welche mit Firebase kommuniziert werden kann
    - Erlaubt es eine Nachricht, in einem Aufruf, an bis zu 500 Geräte zu verschicken

13.3.2 Codeprüfung

Um eine hohe Qualität für den gesamten Source Code zu gewährleisten, werden Entwicklungstools zur Codeüberprüfung benötigt. Zum Einen soll diese mit Hilfe von statischer Codeanalyse stattfinden, um eine frühe Fehlererkennung und die Durchsetzung eines einheitlichen Codestyles zu ermöglichen. Zum Anderen soll die Funktionalität des Codes stetig durch Unit- und Integrationstests überprüft werden, was ermöglicht bei jeder Änderung des Codes die Lauffähigkeit der bestehenden Funktionalitäten sicherzustellen.

Bei der Auswahl der entsprechenden Tools ist es wichtig zu garantieren, dass die Codeprüfung im Rahmen einer Continuous Integration automatisierbar ist.

Verwendete Software

- Frontend
  - Lint (https://pub.dev/packages/lint)
    - Definiert Coderichtlinien gemäss des offiziellen Dart-Styleguides
    - Ermöglicht die Überprüfung der Coderichtlinien innerhalb der IDE
  - Mockito (https://pub.dev/packages/mockito)
    - Bietet Funktionalität an, um Mocks für Dart-Komponenten zu erstellen
  - Dart code metrics (https://pub.dev/packages/dart_code_metrics)
    - Bietet Funktionalität zur statischen Code-Analyse für Dart an
    - Ermöglicht die automatisierte Überprüfung einiger NFRs
- Backend
  - StyleCop Analyzers (https://github.com/DotNetAnalyzers/StyleCopAnalyzers)
    - Definiert Coderichtlinien welche nach Bedarf angepasst werden können
    - Ermöglicht die Überprüfung der Coderichtlinien innerhalb der IDE
  - Menees Analyzers (https://github.com/menees/Analyzers)
    - Definiert zusätzliche Coderichtlinien, betreffend der erlaubten Länge von Klassen/Methoden/Zeilen, welche nach Bedarf angepasst werden können
    - Ermöglicht die Überprüfung der Coderichtlinien innerhalb der IDE
  - Nsubstitute (https://nsubstitute.github.io/)
    - Bietet Funktionalität an, um Mocks für C# Interfaces und auch gewisse Klassen zu erstellen.

13.3.3 Code-Dokumentation

Da das Projekt nach Ende der Bachelorarbeit höchstwahrscheinlich weiterentwickelt bzw. weiterbetrieben wird, ist es elementar die Einarbeitung in das Projekt so einfach wie möglich zu gestalten. Daher wird ein Grossteil des Source Codes dokumentiert.

Verwendete Software

- Frontend
  - Dartdoc (https://dart.dev/tools/dartdoc)
    - Ermöglicht die Dokumentation von Dart (Flutter) Code.
    - Die Dokumentation kann direkt im Source Code mittels spezieller Kommentare und Annotationen erfolgen.
    - Bietet die Generierung der Code-Dokumentation in Form einer interaktiven Webseite an.
- Backend
  - Doxygen (https://www.doxygen.nl/index.html)
    - Ermöglicht die Generierung einer Dokumentation aus den in C# integrierten XML-Kommentaren
    - Ermöglicht verschiedene Ausgabeformate

13.3.4 Unit-Tests

Um eine gute Codequalität zu gewährleisten, muss der Code auch regelmässig getestet werden. Die am besten in die CI integrierbare Art des Testens sind die Unit-Tests. Um sicherzustellen, dass der Code auch das macht, was der Entwickler sich bei der Implementation gedacht hat, werden sowohl im Frontend als auch im Backend Unit-Tests verwendet. Diese helfen ausserdem nach vorgenommenen Änderungen zu prüfen, ob der Code noch erwartungsgemäss funktioniert.

Verwendete Software

- Frontend
  - Flutter Test (https://flutter.dev/docs/cookbook/testing/unit/introduction)
    - Ermöglicht Unit- und Widget-Testing für Flutter
    - Ist direkt im Flutter SDK integriert
  - LCOV (http://ltp.sourceforge.net/coverage/lcov.php)
    - Ermöglicht die Test-Coverage von Flutter Test als interaktive Webseite darzustellen
- Backend
  - Xunit (https://github.com/xunit/xunit)

    - Ist ein Tool, welches das Unit-Testing ermöglicht
    - Tests sind integriert in die IDE und können direkt ausgeführt werden
    - Tests können auch über die Kommandozeile ausgeführt werden
- Coverlet.Collector (https://github.com/coverlet-coverage/coverlet)
    - Ermöglicht es, zu analysieren, wie viel und welcher Code durch Unit-Tests abgedeckt ist

14 Logische Architektur

In diesem Kapitel wird die interne Struktur der verschiedenen Komponenten beschrieben, sowie die Zusammenhänge zwischen den Komponenten dargestellt. Darüber hinaus werden die wichtigsten Abläufe innerhalb der Software erklärt.

14.1 Server

Der Serverteil der Applikation ist in Schichten gegliedert, welche wiederum aus verschiedenen Klassen bestehen. Die nachfolgende Grafik soll einen Überblick über die Architektur des Servers und die wichtigsten Zusammenhänge der verschiedenen Klassen geben. Hierbei ist anzumerken, dass die Grafik sehr vereinfacht ist und viele Implementationsdetails nicht dargestellt werden, um die Übersichtlichkeit zu gewährleisten. Um trotzdem einen guten Überblick zu ermöglichen, wird die Architektur anschliessend textuell erläutert.

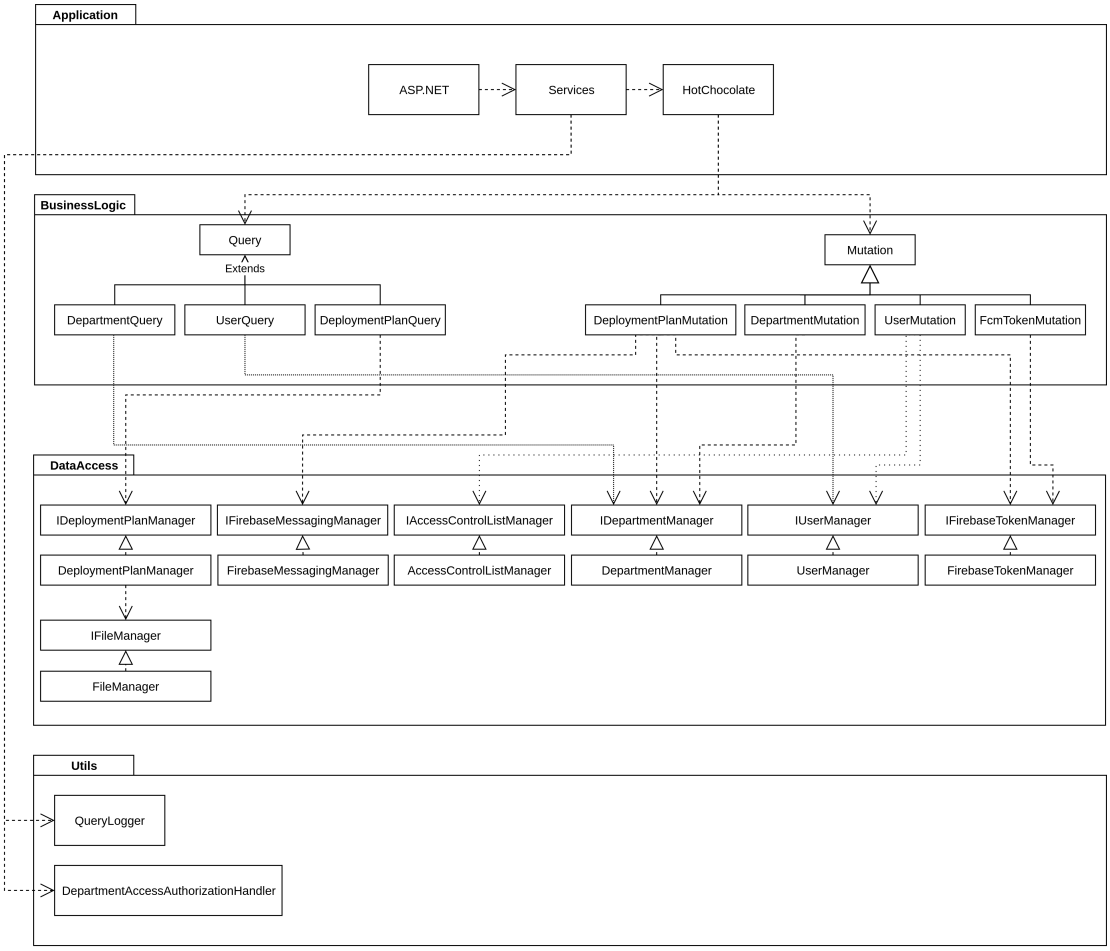

Abbildung 14.1: Schichtenarchitektur des Servers

14.1.1 Application

Auf der Applikationsschicht wird auf ASP.NET core mit dem Kestrel Webserver gesetzt. ASP.NET erlaubt es uns Services zu definieren und zu deklarieren, welche mittels Dependency Injection an andere Services oder Klassen übergeben werden können. Die Anfragen werden, nachdem Sie von ASP.NET empfangen wurden, an HotChocolate weitergeleitet. Bei HotChocolate handelt es sich um die verwendete GraphQL-Server-Implementation. Diese wird bei ASP.NET als Service registriert, analysiert anschliessend alle eingehenden Anfragen und leitet diese entsprechend ihrem Inhalt weiter [26].

14.1.2 Business Logic

In der Business Logic Schicht befinden sich alle API-spezifischen Methoden. GraphQL unterstützt mehrere Operationsarten, wobei „Query“ für Lesen und „Mutation“ für Veränderung steht. Es gäbe noch mehr Operationen, welche in GraphQL definiert sind, diese werden im Server allerdings nicht verwendet und dadurch hier nicht erwähnt.

Nachdem „HotChocolate“ die Anfragen erhalten hat, werden diese entsprechend ihrer Operation weitergeleitet [26]. Die Operationen wurden auf dem Server nach Domain-Objekten aufgeteilt um kleinere übersichtlichere Klassen zu erhalten.

Die Operations-Klassen rufen anschliessend die Manager auf, welche benötigt werden, um eine Operation durchzuführen. Teilweise müssen hierbei mehrere Manager aufgerufen werden, da einige Operationen auf mehrere Domain-Objekte zugreifen müssen.

14.1.3 Data Access

In der Data Access Schicht befinden sich die Manager, welche für jeweils ein Domain-Objekt zuständig sind. Sie überprüfen Voraussetzungen und werfen Fehler, falls diese nicht erfüllt sind oder eine Operation so nicht durchgeführt werden kann. Darüber hinaus greifen sie auf die Persistenz (Datenbank) oder eine externe Schnittstelle (z.B. Firebase) zu. Die Manager werden über ihr Interface bei ASP.NET als Services registriert und anschliessend mithilfe von Dependency Injection mit den Operations-Klassen verknüpft. Somit könnte die Implementation ausgetauscht werden, solange das Interface bestehen bleibt.

14.1.4 Utils

Um die API und die gesamte Serverapplikation umzusetzen sind, neben den bereits vorgestellten, noch einige weitere Komponenten notwendig. Diverse davon können keiner Schicht zugeordnet werden, da Sie als Helfer-Klassen agieren.

14.1.4.1 Autorisierung

Die Authentisierung wird von ASP.NET selbst übernommen. Diese wird mithilfe eines JWT-Token abgewickelt. Die Autorisierung muss allerdings manuell definiert werden, da ASP.NET nicht wissen kann, was für Regeln hier erwünscht sind. Hierzu werden bei den GraphQL-Operationen die Autorisierungs-Policies angegeben, welche verwendet werden sollen und als Service der manuell implementierte Autorisierungs-Handler registriert. Somit kann die Autorisierung über ASP.NET-Bordmittel geschehen, aber trotzdem mit der eigenen Business-Logik gearbeitet werden. Auf die Autorisierung wird in Abschnitt IV noch genauer eingegangen.

```
1  // Es wird mit der Policy "EditUser" gearbeitet.
2  [Authorize(Policy = "EditUser")]
3  // Das Argument "departmentId" dieser Anfrage enthaelt
4  // die id der Abteilung, mithilfe derer die Abteilungs-
5  // abhaengigen Berechtigungen geprueft werden koennen.
6  [AccessObjectIdArgument("departmentId", true)]
7  public async Task<User> CreateUser(
8    ...,
9    string userName,
10   string password,
11   AccessControlList accessControlList,
12   int departmentId){}
```

Auflistung 14.1: Operation mit Autorisierung

Eine weitere Klasse der Utils Schicht ist der Logger. Wir haben zum besseren Logging einen Query-Logger implementiert, welcher alle Anfragen mit der dazugehörigen Query dokumentiert.

14.2 App

Wie der Server, ist auch die App in Schichten gegliedert, die sich wiederum aus Klassen zusammensetzen. Die folgende Grafik soll einen Überblick über die Architektur der App und die wichtigsten Zusammenhänge der verschiedenen Klassen geben. Wie beim Server schon erwähnt, ist auch dieses Schichtendiagramm vereinfacht um die Übersichtlichkeit zu gewährleisten. Relevante Implementationsdetails sind in der nachfolgenden Beschreibung erläutert.

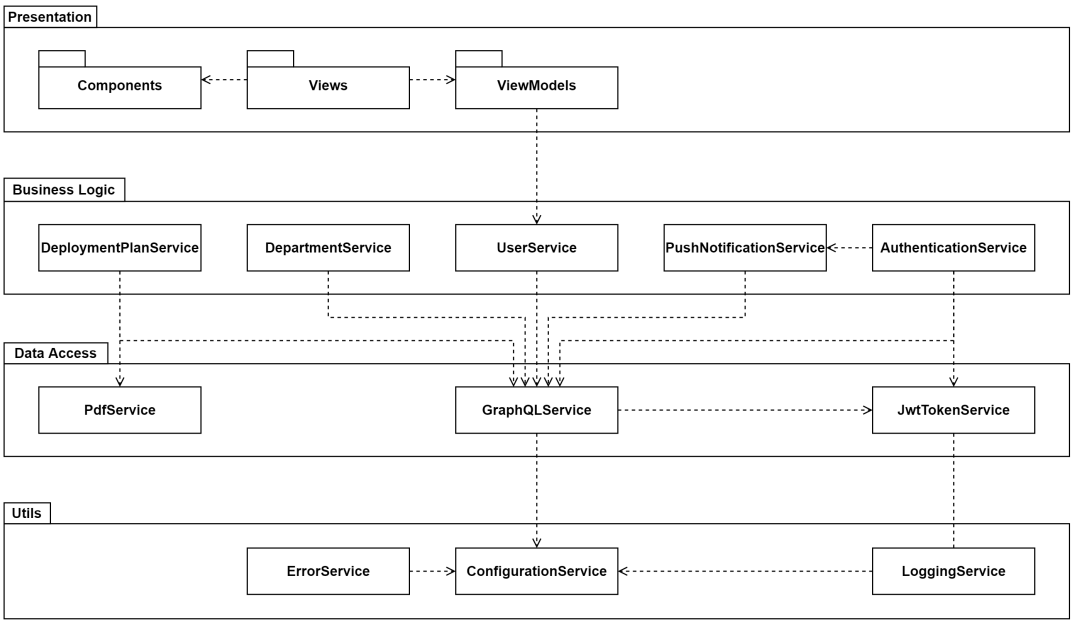

Abbildung 14.2: Schichtenarchitektur der Browsererweiterung

14.2.1 Presentation

Die Presentation Schicht setzt sich aus den Views zusammen, welche zusammen mit den Components und ViewModels die Benutzeroberfläche bilden. Anders ausgedrückt befindet sich in dieser Schicht all der Flutter-spezifische Code. Alle weiteren Schichten der App könnten auch unabhängig von Flutter in anderen Dart-Projekten wiederverwendet werden.

Die Aufgaben der Klassen in der Presentation Schicht ist folgendermassen aufgeteilt. Für jede Ansicht auf der Benutzeroberfläche existiert eine View mit zugehörigen ViewModel. Eine View, welche sich aus mehreren Components zusammensetzt, ist dabei jeweils zuständig für das Rendern der UI-Elemente. Das ViewModel beinhaltet die zugehörige Funktionalität, wie beispielsweise die Formularvalidierung und die Kommunikation mit den Services aus den unteren Schichten. Components können als kleinere Views betrachtet werden, da ihre primäre Zuständigkeit darin liegt, Teile aus Views zur Wiederverwendung und zur Einhaltung des Single-Responsibility-Prinzip zu extrahieren. Die Aufteilung von View und ViewModel existiert bei Components auch, sofern diese Funktionalität beinhalten.

14.2.1.1 Kommunikation mit weiteren Schichten

Wie bereits erwähnt, sind die ViewModels zuständig für die Kommunikation mit den unteren Schichten. Hierbei findet die Kommunikation initial über einen Mittelsmann, den sogenannten Service Locator statt. Dieser ist zuständig für die Verwaltung der Services und deren Lifecycle.

Dies bedeutet, dass initial alle Services auf dem Service Locator registriert werden, um später von den ViewModels abgefragt werden zu können. Sobald ein Service dann von einem ViewModel angefragt wird, konstruiert der Service Locator den Service gemäss Konfiguration und übergibt diesen an das ViewModel zur weiteren Verwendung.

14.2.2 Business Logic

In der Business Logic Schicht befindet sich ein Teil der Services. Diese sind primär, nach den Domain-Objekten mit welchen sie arbeiten, gegliedert. Hierbei gibt es unter anderem einen Service für jedes Domain-Objekt, welcher dafür zuständig ist, Anfragen von den ViewModels entgegenzunehmen und diese entsprechend an die Data Access Schicht weiterzuleiten.

Ein Beispiel für einen solchen Ablauf wäre das Abrufen von allen aktuellen Einsatzplänen. Ein ViewModel aus der Presentation Schicht ruft den Service mit den entsprechenden Parametern auf, dieser arbeitet entsprechend der Anfrage die korrekte GraphQL Query aus und schickt diese an den GraphQLService der Data Access Schicht weiter. Sobald sich dieser mit einer Antwort zurückmeldet, konvertiert der Service in der Business Logic Schicht die Daten in die entsprechenden Business-Objekte um und gibt diese dem anfragenden ViewModel zurück.

14.2.3 Data Access

In der Data Access Schicht befinden sich weitere Services, welche allerdings nicht nach Business Objekten sondern nach zugrundeliegenden Technologien aufgeteilt sind. Diese werden, wie auch bei der Kommunikation zwischen Presentation und Business Logic Schicht, über den Service Locator angesteuert.

Bei diesen Services handelt es sich um den PdfService, welcher statische Dateien vom Server abruft. Zusätzlich gibt es den GraphQLService, welcher mit dem GraphQL Endpoint des Server kommuniziert und zu guter Letzt gibt es den JwtTokenService, welcher zuständig für die Verwaltung des Authentisierungs-Tokens im Speicher der jeweiligen Plattform ist.

14.2.4 Utils

Um die Architektur abzurunden, gibt es noch weitere Komponenten, welche sich nicht in die anderen Schichten eingliedern lassen. Dabei geht es um Funktionalitäten, welche über alle Schichten gebraucht werden. Dies sind beispielsweise Error Handling, Konfiguration und Logging.

15 Schnittstellen

In diesem Kapitel werden die Schnittstellen beschrieben, welche die Applikation nutzt. Ausserdem wird die vom Server selbst angebotene Schnittstelle, die GraphQL-API, visualisiert.

15.1 Server

Der Pronto-MIA-Server benutzt selbst eine Schnittstelle und bietet wiederum eine API-Schnittstelle an.

15.1.1 Firebase Messaging Service

Der „Firebase Messaging Service“ ist die externe Schnittstelle welche der Server verwendet. Der Service wird genutzt um Push-Benachrichtigungen an die entsprechenden Geräte der Benutzer zu verteilen. Hierzu wird der Endpunkt „SendMulticastAsync“ der Firebase-Messaging-API verwendet [27].

15.1.2 API

Wir haben die Technologie GraphQL unter anderem gewählt, da durch das Schema die API dokumentiert werden kann. Alle Endpunkte und Parameter der API wurden dokumentiert und diese Dokumentation kann durch das Abrufen des Schemas direkt angeschaut werden. Im Anhang H wurde das Schema der API abgelegt, welches alle Informationen der API beinhaltet. Zusätzlich wird nachfolgend eine vereinfachte Visualisierung aller Endpunkte dargestellt.

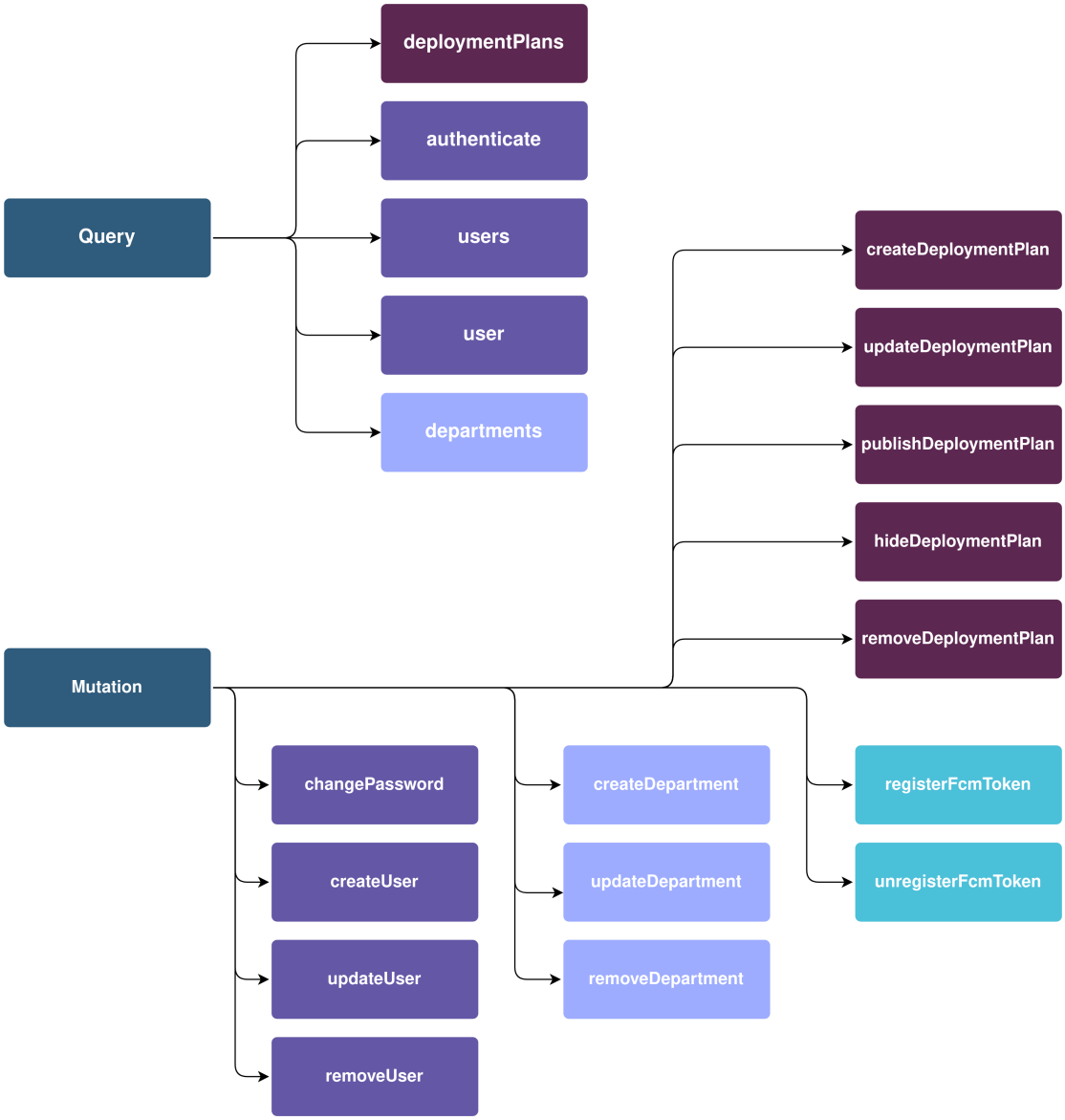

Abbildung 15.1: Übersicht über die GraphQL-API

15.2 App

Das Pronto-MIA-App kommuniziert mit zwei verschiedenen Schnittstellen. Zum Einen mit der zuvor definierten GraphQL-API des Servers, wobei eine GraphQL-Library zur Kommunikation über HTTP(S) verwendet wird. Zum Anderen kommuniziert das App, wie der Server auch, mit dem Firebase Messaging Service.

Für die Kommunikation zwischen der App und dem Firebase Messaging Service sind zwei verschiedene Libraries notwendig. Es wird die Firebase Core Library benötigt, welche die grundsätzliche Kommunikation mit Firebase ermöglicht. Zusätzlich wird allerdings noch die Firebase Messaging Library benötigt, um auf Messaging-spezifische Funktionen zugreifen zu können.

Das App benötigt hierbei Funktionalität um den Benutzer nach Berechtigungen zum Empfang von Push-Benachrichtigungen zu fragen, eindeutige Tokens für Geräte von Firebase zu holen und sich für verschiedene Events zu registrieren. Diese Events ermöglichen den Empfang von Benachrichtigungen, auch wenn die App nicht läuft.

16 Continuous integration

Um eine gute Code-Qualität zu garantieren, wird CI verwendet. Es ermöglicht uns den, in der Versionverwaltung eingecheckten, Code permanent zu überprüfen. Die CI-Pipeline wurde im Rahmen dieses Projektes mithilfe von Github-Actions umgesetzt. Nachfolgend sind die Abläufe der jeweiligen Pipelines abgebildet. Dabei zu beachten ist, dass jeder Schritt eine erfolgreiche Durchführung des Schritts davor voraussetzt.

App

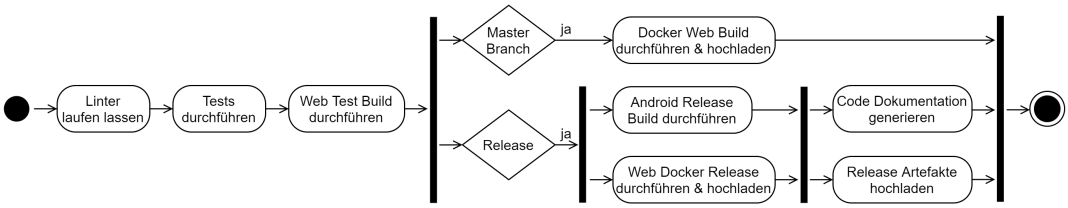

Abbildung 16.1: CI der App

Server

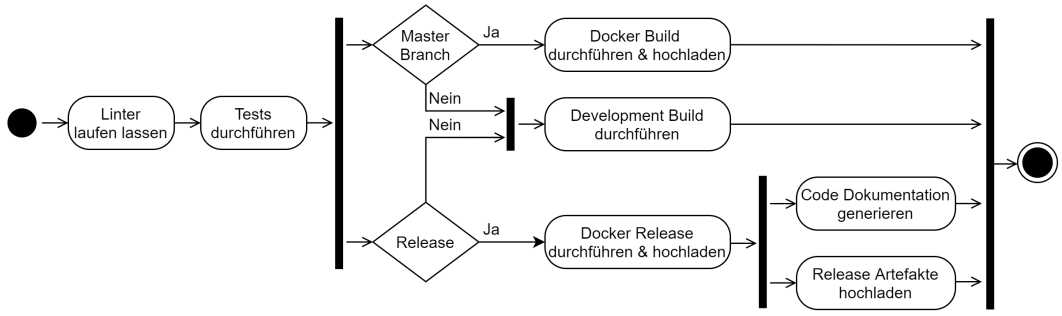

Abbildung 16.2: CI des Servers

17 End of Elaboration

Das nachfolgende Kapitel befasst sich mit der Auflösung von Unklarheiten und Unsicherheiten. Dies erfolgt aus technischer Sicht durch die Erstellung eines Prototypen, in welchem die wichtigsten Funktionen rudimentär implementiert werden um die technische Machbarkeit des Projekts sicherzustellen. Darüber hinaus wird dies aus administrativer Sicht durch die Prüfung der End of Elaboration Checklist, gemäss der SE1-Vorlesung, sichergestellt.

17.1 Technischer Prototyp

Das Ziel des technischen Prototyps ist es sicherzustellen, dass das Projekt mithilfe der evaluierten Technologien durchgeführt werden kann. Dafür wurden unter Berücksichtigung der Anforderungsspezifikation die nachfolgenden Features ausgewählt, welche betreffend der Implementation eine Herausforderung darstellen könnten:

- Kommunikation zwischen Client und Server mittels der GraphQL-API.
- Benutzer-Authentifikation über die GraphQL-API.
- Hoch- und Herunterladen von PDF-Dateien über die GraphQL-API.
- Anzeige von PDF-Dateien innerhalb der App auf allen Zielplattformen.
- Versenden und Empfangen von Push-Benachrichtigungen.

Aufgrund dieser Eigenschaften wurde entschieden, den Prototyp in Form einer App zu gestalten, welche die Möglichkeit bietet eine PDF-Datei hochzuladen. Anschliessend soll die App die Datei wieder vom Server herunterladen, um dem Benutzer die Möglichkeit zu geben, diese anzusehen. Zusätzlich soll noch ein simples Login-Formular implementiert werden, welches die vorher genannte Funktionalität schützt.

17.2 Checkliste

Nach der erfolgreichen Implementation des technischen Prototyps, wird zur bereits erwähnten Checkliste Stellung genommen:

- Wir haben den Kunden verstanden (Requirements so vollständig wie nur möglich). Scope (=grober Funktionsumfang) ist vereinbart.
  - Die Anforderungen des Kunden (Pronto AG) sind in der Aufgabenstellung aufgeführt.

  - In der Anforderungsspezifikation´ (8) wurden diese über mehrere Iterationen spezifiziert und erweitert.
  - Die finale Anforderungsspezifikation wurde vom Kunden und der Betreuungsperson überprüft und abgenommen.

- Wir haben alle Werkzeuge im Griff (IDE, Versionskontrolle, Build Server, Deployment, Unit Testing, Workflow Tools, Wiki, ...).
  - Sämtliche Repositories wurden inkl. CI/CD aufgesetzt.
  - Die Entwicklungstools wurden im Rahmen des technischen Prototyps bei allen Entwicklern eingerichtet und überprüft.
  - Das Deployment des Backends, sowie der Webversion wurde im Rahmen des technischen Prototyps bereits getestet.
  - Das erstmalige Deployment der App in die App Stores von iOS und Android wird nach der Fertigstellung des ersten Releases durchgeführt.
- Wir wissen, wie die Architektur aussehen wird (Architektur skizziert, die grossen Interfaces festgelegt, Architektur-Prototypen gemacht).
  - Aufgrund fehlender Erfahrung mit den verwendeten Technologien wurde darauf verzichtet, die Architektur vorgängig zu skizzieren.
  - Die grobe Architektur wurde im Rahmen des technischen Prototyps ausgearbeitet.
  - Die aus der Arbeit resultierende Architektur ist im Abschnitt III dokumentiert.
- Für das User Interface gibt es einen ersten Entwurf (Grafiken, Wireframes, klickbarer Prototyp), der dem Kunden gefällt.
  - Der Entwurf des User Interfaces (8.4.2) wurde während der Anforderungsspezifikation zusammen mit dem Kunden ausgearbeitet.
  - Die finale Version des Entwurfs wurde mit dem Kunden besprochen und wurde entsprechend abgenommen.
- Wir haben die Marschrichtung (grobe Meilensteine) für die nächsten zwei Iterationen festgelegt, viele Arbeitspakete erstellt, und dem Kunden eine genauere Zeitschätzung geliefert.
  - Der Zeitplan wurde im Projektplan (I) definiert.
  - Die grobe Marschrichtung wird durch den Releaseplan (8.5) vorgegeben.
  - Die Arbeitspakete für den nächsten Sprint wurden in Form von GitHub Issues definiert.

- Alle grossen Risiken und Fragezeichen sind weg.
  - Die Anforderungen wurden zusammen mit dem Kunden so weit wie möglich spezifiert.
  - Der erfolgreiche Abschluss des technischen Prototyps gilt als Beweis für die technische Machbarkeit des Projekts.
  - Mögliche Risiken wurden in Kapitel 6 zeitlich geschätzt und eingeplant.

Teil IV

Umsetzung

18 Herausforderungen

Während der Umsetzungsphase, eines Projektes, treten oftmals unvorhergesehene Situationen ein. Dies zeigt sich zumeist durch ein unerwartetes Ausmass an Komplexität oder benötigtem Zeitaufwand für die Implementation einer bestimmten Funktionalität. Diese Herausforderungen werden hier dokumentiert, um die Erfahrungen und getesteten Lösungswege festzuhalten.

18.1 Frontend

In diesem Kapitel werden die Herausforderungen beschrieben, welche bei der Implementation des Frontend aufgetreten sind. Es wurde jene Herausforderungen dokumentiert, welche während der Implementation besonders viel Aufmerksamkeit benötigten.

18.1.1 Navigation

Flutter bietet, für den Wechsel zwischen verschiedenen Ansichten auf der Benutzeroberfläche, mehrere Möglichkeiten der Navigation an.

Der erste Ansatz, auch als „Navigator 1.0“ bekannt beinhaltet, dass der `Navigator` Widgets oben auf einen Stack legt, wobei das oberste Element die aktuelle Ansicht darstellt. Hierbei muss das jeweilig anzuzeigende Widget gemäss nachfolgendem Beispiel, innerhalb eines `Route`-Objekts, übergeben werden.

```
1 Navigator.push(
2   context,
3   MaterialPageRoute(
4     builder: (context) {
5       return StartupView();
6     },
7   ),
8 );
```

Auflistung 18.1: Navigation mittels Navigator 1.0

Ein entscheidender Nachteil dieses Ansatzes ist es, dass bei jeder Navigation der aktuelle `BuildContext` mitgegeben werden muss, welcher dem App vom Framework übergeben wird. Diese Bedingung führt dazu, dass es umständlich werden kann, in Komponenten ausserhalb von Widgets die Ansicht zu wechseln. Grund dafür ist, dass diese normalerweise keinen Zugriff auf den `BuildContext` erhalten.

Diese Einschränkung wird beispielsweise dann sichtbar, wenn in einer globalen Fehlerbehandlung festgestellt wird, dass die Sitzung des aktuellen Benutzers abgelaufen ist und er sich daher neu anmelden muss. Hierzu müsste aus der Fehlerbehandlung heraus, die aktuelle Ansicht auf das Login-Formular geändert werden. Da die Fehlerbehandlung keinen zugriff auf den `BuildContext` hat entsteht hier ein Problem.

Eine etwas neuere Methode bildet die sogenannte „Navigator 2.0 API“, wobei es sich um einen deklarativen Ansatz handelt. Dabei muss die Entscheidungslogik, wann und welches Widget gerendert wird, manuell implementiert werden. Vorteilhaft dabei ist die integrierte Möglichkeit auf Informationen der zugrundeliegenden Plattformen, wie beispielsweise die URL im Web, zugreifen zu können [28].

Die Stacked Library, welche bereits aufgrund der Integration von ViewModels im Projekt verwendet wird, bietet eine Möglichkeit an, die Vorteile beider Ansätze zu verbinden. Sie implementiert einen Service, bei welchem ohne `BuildContext` mit einem Stack navigiert werden kann. Dies erfolgt, ohne alle Routen und die zugehörige Entscheidungslogik definieren zu müssen, wie es aus dem „Navigator 1.0“-Ansatz bekannt ist. Unglücklicherweise ist es damit aber lediglich möglich die Browser-URL zu lesen und danach zu navigieren, nicht aber nach einer erfolgten Navigation diese zu aktualisieren. Somit kann, in der Web-Implementation der App, die URL nicht kopiert werden um eine Ansicht wieder aufzurufen. Der Ansatz bietet des Weiteren die Möglichkeit, ohne weiteren Aufwand, Übergänge in die Navigation einzubinden.

Aufgrund der zuvor genannten Fakten wurde entschieden, den `NavigationService` der Stacked Library zu verwenden. Sollte vom Kunden her die Anforderung aufkommen, in der Webversion auch über URLs navigieren zu können, müsste dies entsprechend auf Anfrage in der Stacked Library implementiert werden oder man müsste die Navigation auf den „Navigator 2.0“-Ansatz umstellen. Beides wäre allerdings mit zusätzlichem Aufwand verbunden.

18.1.2 App-Konfiguration

Flutter bietet nach momentanem Stand keine Möglichkeit an, um eigene App-spezifische Einstellungen zu definieren. Aus diesem Grund wurde während der Umsetzung eine entsprechende Library evaluiert, welche diese Aufgabe übernimmt. Die Anforderungen an eine solche Library wurden folgendermassen definiert:

- Die Library muss auf allen Zielplattformen (Android, iOS und Web) funktionieren.
- Es sollen mehrere Konfigurationsdateien erstellt werden können, um mehrere Umgebungen (Produktion/Entwicklung) abbilden zu können.

Anhand der Kriterien fiel die Wahl auf Flutter „Global Configuration“ (`https://pub.dev/packages/global_configuration`). Diese Library ermöglicht es, Konfigurationsdateien im JSON-Format zu erstellen. Diese werden anschliessend als herkömmliche Flutter-Assets registriert, damit sie mit dem App paketiert werden. Die Library erlaubt es hierbei mehrere Konfigurationsdateien zu laden, wodurch eine strikte Trennung von Entwicklungs- und Produktivkonfiguration ermöglicht wird.

Innerhalb der Pronto-MIA-App wurde ein Wrapper für die Library als Singleton-Service implementiert. Dies ermöglicht den Zugriff auf die Konfiguration über das gesamte App. Nachfolgend wird das Interface des Services dargestellt:

```
1 abstract class ConfigurationService {
2   // laedt alle noetigen Konfigurationsdateien
3   Future<void> init() async;
4
5   // Ermoeglicht das Abfragen von Konfigurationsoptionen
6   T getValue<T>(String key);
7 }
```

Auflistung 18.2: Interface des ConfigurationService

18.1.3 Fehlerbehandlung

Flutter bietet standardmässig keine Möglichkeit an, einheitliche Fehlerbehandlung zu betreiben. Hier schafft die verwendete Library „Stacked“ jedoch Abhilfe, mit der Funktion `runBusyFuture`. In allen verwendeten ViewModels ist es möglich, diese Funktion auszuführen. Tritt in der darin enthaltenen Funktion ein Fehler auf, ungeachtet davon ob dieser asynchroner oder synchroner Natur ist, wird dieser abgefangen. Der Fehler ist anschliessend als Attribut des ViewModels über `modelError` abrufbar und dessen Existenz kann mittels `hasError` überprüft werden.

ViewModels die bei ihrer Initialisierung Daten laden, wie es beispielsweise bei `FutureViewModel` der Fall ist, gehen betreffend dieser vereinheitlichten Fehlerbehandlung einen Schritt weiter. Sie ermöglichen die Implementation eines Handlers für das `onError`-Event, welches ausgeführt wird sollten Fehler beim Laden eben dieser Daten auftreten.

Aufgrund dieser Möglichkeiten, hat sich über die gesamte Applikation folgender Ablauf zum generellen Abfangen von Fehlern etabliert:

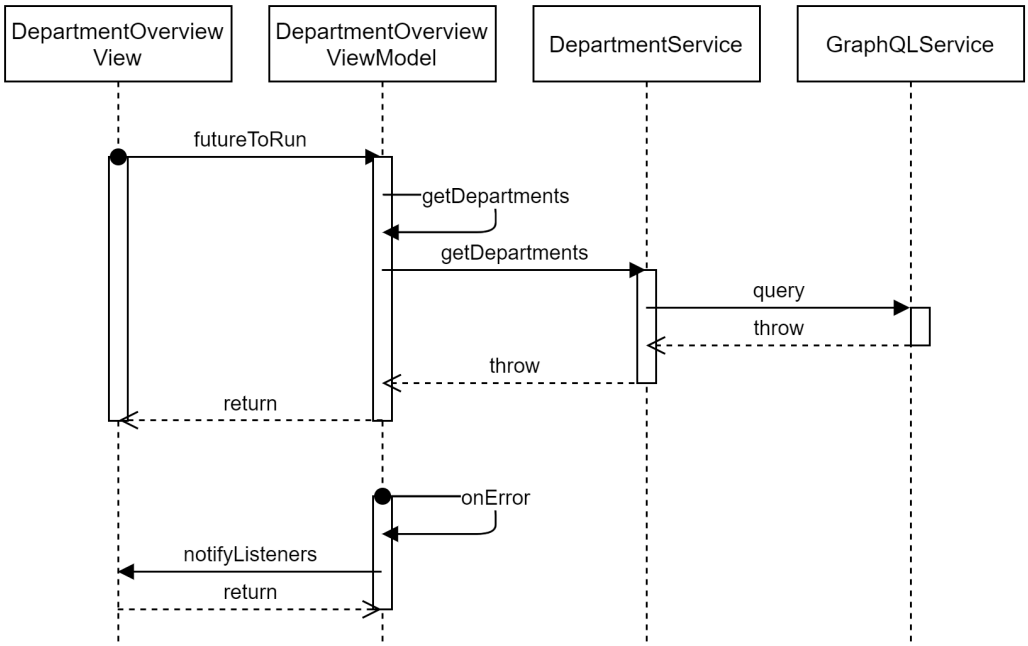

Abbildung 18.1: Fehlerbehandlung bei initialem Laden der Abteilungsliste

18.1.4 Asynchrone Services

Eine Kernfunktionalität der Library „Stacked“ ist es, Services über einen Service Locator zu registrieren. Dies erfolgt standardmässig über Annotationen. Aus diesen wird anschliessend mithilfe der zusätzlichen Library „Build Runner“ der Service Locator generiert [29]. Ein Beispiel für eine solche Konfiguration könnte wie folgt aussehen:

```
@StackedApp(
  dependencies: [
    LazySingleton(classType: PrimaryService, asType:
       PrimaryService),
    LazySingleton(classType: SecondaryService, asType:
       SecondaryService),
  ],
)
```

Auflistung 18.3: Annotationen zur Generierung des Service Locator

Während der Umsetzung des Projekts hat sich gezeigt, dass einige Services eine initiale Konfiguration ihrer zugrundeliegenden Funktionalitäten benötigen. Bei einigen davon ist dies nur durch asynchronen Code möglich. Dies führt dazu, dass diese initiale Konfiguration nicht mehr über den Konstruktor erfolgen kann. Dies da es nicht möglich ist, den Konstruktor als asynchron zu definieren.

Ein Beispiel für solch einen Service ist der `ConfigurationService`. Damit er verwendet werden kann ist es nötig, initial alle benötigten Konfigurationsdateien einzulesen. Dies erfolgt über die asynchrone Methode `loadFromAsset` der Library „Global Configuration“.

Ein erster Ansatz diesem Problem entgegenzuwirken, war es, die initiale Konfiguration der betroffenen Services während des Splash-Screens `StartupView` durchzuführen. Dies hat sich mit der Zeit aber aus zwei Gründen als ungeeignet herausgestellt. Einerseits wird der Splash-Screen und das zugehörige ViewModel nicht bei jedem App-Start aufgerufen. Dies führt dann dazu, dass Services nicht richtig konfiguriert sind, sobald sie das erste Mal verwendet werden sollen. Des Weiteren verzögert diese Methode den Startvorgang, da einige Services initial konfiguriert werden müssen. In der aktuellen App-Version hat sich dies performancetechnisch zwar noch nicht bemerkbar gemacht, aber es wäre gut möglich, dass dies in Zukunft problematisch werden könnte.

Um diese Herausforderung zu lösen, wurde entschieden, den Service Locator manuell zu implementieren. Die generierte Variante aus den Annotationen konnte hierbei als Vorlage benutzt werden. Die manuelle Implementation erlaubt es, bei der Registrierung des Services eine asynchrone Methode mitzugeben. Diese wird dann vom Service Locator aufgerufen, um den Service zu initialisisieren:

```
1  final locator = StackedLocator.instance;
2
3  void setupLocator() {
4    locator.registerLazySingletonAsync<ConfigurationService>(()
        async {
5      final configurationService = ConfigurationService();
6      await configurationservice.init();
7      return configurationService;
8    })
9
10   // ...
11 }
```

Auflistung 18.4: Manuelle Implementation des Service Locator

18.1.5 Hochladen von Dateien

Während der Implementation des Formulars, zum Hochladen von Dateien, im technischen Prototyp sind zwei Herausforderungen in verschiedenen Abschnitten des Prozesses zum Vorschein gekommen. Diese werden nachfolgend dokumentiert.

18.1.5.1 Unterschiede zwischen den Plattformen

Für die Auswahl von Dateien über die Benutzeroberfläche wurde auf die Library „File Picker“ (`https://pub.dev/packages/file_picker`) zurückgegriffen. Diese ermöglicht es, unter anderem, über `FilePicker.platform.pickFiles()` die native Dateiauswahl der jeweiligen Plattform zu öffnen. Hierbei wird nach erfolgreicher Auswahl einer Datei ein `FilePickerResult`-Objekt zurückgegeben.

Problematisch dabei ist allerdings die Tatsache, dass für das darin enthaltene `PlatformFile` je nach Plattform andere Attribute gesetzt werden. Konkret heisst dies, dass auf den Mobile-Plattformen, wie Android und iOS, der Dateiinhalt über das Attribut `path` abrufbar ist und bei Web über `bytes`. Zurückzuführen ist diese Problematik auf die zugrundeliegenden Berechtigungen der einzelnen Plattformen. Im Falle des Webs hat der Browser nämlich keinen Zugriff auf den Dateipfad der, in den Browser hochgeladenen, Datei.

Bewältigt wurde diese Herausforderung durch das Anhängen folgender Logik an den Handler der Dateiauswahl:

```
SimpleFile fileUpload;

if (kIsWeb) {
  fileUpload = SimpleFile(
    name: result.names.single,
    bytes: result.files.single.bytes,
  );
} else {
  final file = File(result.files.single.path);
  fileUpload = SimpleFile(
    name: result.names.single,
    bytes: file.readAsBytesSync(),
  );
}
```

Auflistung 18.5: Vereinheitlichung von hochzuladenden Dateien über alle Plattformen

Mit der Hilfe dieses Codes wird die Datei plattformabhägig in ein `SimpleFile`-Objekt konvertiert, bei welchem der Dateiinhalt in allen Fällen über das Attribut `bytes` abgefragt werden kann. Es gilt zu beachten, dass für die mobilen Plattformen aus dem Dateipfad erst noch ein `File`-Objekt erstellt werden muss, bevor dieses anschliessend in Bytes umgewandelt werden kann. Dieser Ansatz ermöglicht es, dass sich im Rest der Applikation nicht mehr um die Unterscheidung verschiedener Plattformen, betreffend hochgeladener Dateien, gekümmert werden muss.

```
class SimpleFile {
  final String name;
  final Uint8List bytes;

  SimpleFile({this.name, this.bytes});
}
```

Auflistung 18.6: Implementation von SimpleFile

18.1.5.2 GraphQL Caching

Im späteren Verlauf der Entwicklung hat sich im Hochlade-Prozess eine weitere Herausforderung aufgezeigt. Es handelt sich dabei um einen Fehler in der verwendeten GraphQL-Library „GraphQL Client“. Beim Hochladen von Dateien über GraphQL gemäss „GraphQL multipart request specification“ wird die hochzuladende Datei zuerst in ein `MultipartFile`-Objekt konvertiert und dann der GraphQL-Query als Variable übergeben [25]. Ein Beispiel hierfür könnte folgendermassen aussehen:

```
final multiPartFile = MultipartFile.fromBytes(
  'upload',
  pdfFile.readAsBytesSync(),
  filename: pdfFile.path,
  contentType: MediaType("application", "pdf"),
);

const query = """
  mutation uploadFile(\$file: Upload!) {
    uploadFile(file: \$file)
  }
""";

final mutationOptions = MutationOptions(
  document: gql(query),
  variables: {"file": multiPartFile},
);
```

Auflistung 18.7: Integration einer Datei in eine GraphQL-Query

Bei der Implementation dieser Mutation wurde festgestellt, dass im `QueryResult`-Objekt, welches, unter anderem, die Antwort vom Server enthält, jedesmal der Fehler `Converting object to an encodable object failed: Instance of 'MultipartFile'` aufgeführt war. Der Fehler wurde jedoch nicht vom Server übergeben, da dieser die Datei wie erwartet entgegengenommen hat und eine entsprechende Erfolgsmeldung zurückgab.

Nach einigem Analyseaufwand wurde festgestellt, dass der Fehler von „GraphQL Client“ stammt und ausgelöst wird, da die Library standardmässig alle Queries und Mutations versucht zu cachen. Aufgrund der fehlenden Implementation von `toJson()` in der “MultipartFile„-Klasse ist dies für solche Dateien allerdings nicht möglich und es wird immer ein Fehler zurückgegeben, da das Caching nicht gemacht werden kann. Problematisch ist zusätzlich, dass neben der Fehlermeldung die eigentliche Antwort des Servers nicht an den Client übergeben wird. Normalerweise könnte dies behoben werden, indem in der Konfiguration der “GraphQL Client„-Library die entsprechende `ErrorPolicy` auf `ignore` oder `all` gesetzt wird. Diese Massnahme hat in diesem Fall jedoch nicht zum Erfolg verholfen.

Das Problem zu umgehen ist dennoch möglich, indem man den internen Cache der “GraphQL Client„-Library deaktiviert. Glücklicherweise kann dies folgendermassen spezifisch für Mutationen gemacht werden:

```
final graphQLClient = GraphQLClient(
  link: link,
  cache: GraphQLCache(),
  defaultPolicies: DefaultPolicies(
    mutate: Policies(
      fetch: FetchPolicy.noCache,
    ),
  ),
);
```

Auflistung 18.8: GraphQl Client ohne Mutations-Cache

Trotz des möglichen Workarounds wurde entschieden, dem Fehler weiter auf den Grund zu gehen. Dafür wurde auf dem GitHub-Repository der „GraphQL Client“-Library ein Issue erstellt, welches die obige Erläuterung noch mit weiteren Implementationsdetails ausführt. Das Issue ist unter https://github.com/zino-app/graphql-flutter/issues/871 zu finden. Unglücklicherweise hat sich bis zum Stand vom 11.06.2021 noch niemand zu der Problematik geäussert.

18.1.6 Änderung der Berechtigungen

Damit das App ohne Einschränkungen funktioniert, ist die Zustimmung des Benutzers zum Empfangen von Push-Benachrichtigungen notwendig. Die zugehörige Anfrage erfolgt beim Start der App bzw. dann wird überprüft ob die Berechtigung bereits vorhanden ist. Sollte dies der Fall sein, wird die Anfrage nicht gestellt. Bei erteilter Berechtigung wird, nach einer erfolgreichen Anmeldung durch den Benutzer, der Token des aktuellen Geräts auf dem Server registriert.

Die Problematik bei diesem Vorgehen ist, dass diese Berechtigung auch ausserhalb der Applikation, über die Einstellungen der jeweiligen Plattform, erteilt oder auch wieder entfernt werden kann. Unglücklicherweise gibt es für keine der Zielplattformen eine Möglichkeit, auf die Änderung der Berechtigungen direkt zu reagieren.

Um dennoch so schnell wie möglich auf eine Änderung der Berechtigungen reagieren zu können, wurde im App ein Handler registriert, der bei jedem Wechsel im Lifecycle des Apps den Status der Berechtigung überprüft und entsprechend das aktuelle Gerät registriert oder aus den registrierten Geräten des Benutzers entfernt. Ein Beispiel für einen solche Statusänderung im Lifecycle wäre, wenn der Benutzer von der geöffneten App in die Geräte-Einstellungen wechselt und dann die Berechtigung ändert. Navigiert er anschliessend wieder zurück zur Pronto-MIA-App, bemerkt diese den Berechtigungswechsel und reagiert entsprechend. Implementiert wurde dies wie folgt:

```
@override
Future<void> didChangeAppLifecycleState(AppLifecycleState
    state) async {
  super.didChangeAppLifecycleState(state);

  if (state == AppLifecycleState.resumed) {
    final isAuthenticated = await
        _authenticationsService.isAuthenticated();
    final notificationsAuthorized = await (await
        _pushNotificationService).notificationsAuthorized();

    if (isAuthenticated && notificationsAuthorized) {
      await (await
          _pushNotificationService).enableNotifications();
    } else {
      await (await
          _pushNotificationService).disableNotifications();
    }
  }
}
```

Auflistung 18.9: Überprüfung der Benachrichtigungs-Einstellungen im Lifecycle

Diese Implementation ist jedoch nicht funktionsfähig für Web, da das `ChangeAppLifecycleState` Event auf dieser Plattform nie ausgeführt wird. In der Webversion ist es somit erforderlich, die Webseite neu zu laden um auf Änderungen an dieser Berechtigung reagieren zu können.

18.1.7 Verschiedene Eingabemethoden

Seit dem Release von Flutter 2 wird Web offiziell als Plattform unterstützt. Trotz dieser Tatsache hat sich während der Implementation einiger Features bemerkbar gemacht, dass Flutter nicht in erster Linie für Web und somit nicht für die Verwendung mit Maus und Tastatur konzipiert wurde. Zwei konkrete Beispiele für diese Herausforderung werden nachfolgend dokumentiert.

18.1.7.1 Fokus von Eingabefeldern

Bei herkömmlichen Webanwendungen und auch Benutzeroberflächen anderer Art verhalten sich Eingabefelder folgendermassen: Wenn ein Benutzer mit der Maus auf ein Eingabefeld klickt, wird dieses fokusiert, was eine Eingabe durch die Tastatur ermöglicht. Klickt der Benutzer anschliessend an einen beliebigen anderen Ort, verliert das aktuelle Eingabefeld den Fokus, womit die Möglichkeit der Eingabe verloren geht.

Flutter ermöglicht grundsätzlich das Fokussieren von Eingabefeldern über die Maus, jedoch ist es nicht möglich, über einen beliebigen Mausklick ausserhalb des Feldes, den Fokus auf ein Eingabefeld wieder zu entfernen.

Dieses unerwartete Verhalten lässt sich glücklicherweise global beheben. Verpackt man das Root-Widget der App (im Falle von Pronto-MIA `MaterialApp`) in einen `GestureDetector`, ermöglicht dieser, die Reaktion auf einen beliebigen Mausklick auf

der Webseite. Entzieht man dann in diesem Handler allen Widgets den Fokus, so führt dies zum erwarteten Verhalten. Aufgezeigt wird dies im nachfolgenden Code-Beispiel:

```
@override
Widget build(BuildContext context) {
  return GestureDetector(
    onTap: () {
      FocusScope.of(context).requestFocus(FocusNode());
    },
    child: MaterialApp(
      // ...
    )
  )
}
```

Auflistung 18.10: Globale Entfernung des Fokus von Eingabefeldern

18.1.7.2 Abschicken von Formularen

In den meisten Webanwendungen ist es möglich, Formulare auf mehrere Arten abzuschicken. Dies kann über den Mausklick auf einen „Abschicken“-Knopf erfolgen oder auch über die Betätigung der Enter-Taste, nachdem alle Eingaben getätigt wurden. Das Abschicken von Formularen mittels der Enter-Taste wird dabei in Flutter nicht standardmässig unterstützt.

Möchte man diese Variante trotzdem unterstützen, setzt dies vorraus, dass Eingabefelder im entsprechenden Formular mithilfe des `TextFormField`-Widgets erstellt wurden. Im Normalfall würde hingegen das `TextField`-Widget verwendet werden. Dadurch wird die Verwendung des Events `onEditingComplete` ermöglicht, in welchem für jedes Formularfeld jene Funktion ausgeführt werden soll, welche das Formular abschickt. Hierbei gilt es zu beachten, dass in dieser Funktion auch immer die Formularvalidierung durchgeführt werden muss, um inkonsistentes oder falsches Verhalten zu vermeiden.

```
Form(
  key: GlobalKey<FormState>(),
  child: Column(
    children: [
      TextFormField(
        controller: userNameController,
        onEditingComplete: model.submitForm,
        decoration: const InputDecoration(labelText:
            'Benutzername *'),
      ),
      TextFormField(
        controller: passwordController,
        onEditingComplete: model.submitForm,
        obscureText: true,
        decoration: const InputDecoration(labelText:
            'Passwort *'),
      ),
      ElevatedButton(
        onPressed: model.submitForm,
```

```
18            child: Text('Anmelden'),
19          ),
20        ],
21      ),
22    )
```

Auflistung 18.11: Beispielformular mit onEditingComplete

18.1.8 PDF Ansicht

Zur Integration von PDF-Dateien in die Benutzeroberfläche, werden in Flutter einige Libraries angeboten. Bei der Evaluation einer solchen Libary haben sich zwei Herausforderungen ergeben.

Einerseits bieten viele dieser Libraries keine Unterstützung für Web an. Dies liegt daran, dass Flutter 2, welches Web als offizielle Plattform eingeführt hat, erst seit einigen Monaten verfügbar ist. Es zeichnet sich ab, dass bei vieler dieser Libraries an der Unterstützung für Web gearbeitet wird.

Andererseits ist die Unterstützung für Web bei jenen Libraries welche es als Plattform unterstützen auch nicht ideal. Dies zeichnet sich durch fehlende Interaktionen mit der Maus ab. Beispielsweise ist es nicht möglich, in der PDF-Ansicht mit dem Mausrad zu scrollen oder auch in Kombination mit der Ctrl-Taste zu zoomen.

Aufgrund dieser Einschränkungen und durch die Tatsache, dass es momentan keine Library zu geben scheint, welche diese Probleme nicht hat, wurde entschieden, PDFs in der Web-Variante anders zu behandeln. Anstatt eine Ansicht zu öffnen, welche das PDF innerhalb der App anzeigt, wird in diesem Fall lediglich das PDF im Browser heruntergeladen und direkt in einem neuen Tab geöffnet.

18.1.9 Sicherheit

In diesem Kapitel werden die beiden Sicherheitsrelevanten Herausforderungen bei der Implementierung des Frontends erläutert. Da die Sicherheit ein wichtiger Bestandteil jeder Applikation ist, wurde bei diesen Herausforderungen besonders gründlich gearbeitet.

18.1.9.1 Sicherheit JWT-Token

Da in der Applikation ein JWT-Token verwendet wird, um den anfragenden Benutzer bei der API zu authentifizieren, muss dieser in der App gespeichert und verwaltet werden. Eine Frage welche sich hierbei stellte war, wie der Token sicher in der App abgelegt werden kann. Initial wurde dieser in den herkömmlichen Key-Value Ablagen der jeweiligen Mobile-Betriebssystemen und im „LocalStorage“ des Browsers abgelegt, allerdings ist keine dieser Ablagestrukturen sicher genug für solch sensitive Daten.

Die Mobilgerät-Betriebssysteme Android und iOS bieten für solch sensitive Daten einen speziell gesicherten Speicher an. Glücklicherweise gibt es für die Verwendung dieses Speichers ein Paket, welches in Flutter eingebunden werden kann und die Verwendung dieses Speichers intuitiv handhabt. Somit konnte der Token sicher abgelegt werden.

Im Web kann das Problem nicht so einfach behoben werden, denn leider gibt es für den Browser keinen solchen sicheren Speicher, in welchem sensitive Daten abgelegt werden

können. Da der Datenspeicher „LocalStorage“ anfällig für Angriffe von böswilligen Javascript-Dateien ist, wurde nach einer Alternative gesucht [30]. Nach einer Analyse, wurde evaluiert, dass es die beste Möglichkeit wäre, einen JWT-Token innerhalb eines Cookies zu speichern, welches nicht von JavaScript ausgelesen werden kann. Hierbei wäre es ausserdem notwendig einen zusätzlichen Token zu implementieren, welcher die Fälschung von Anfragen verhindert [31].

Aufgrund dieser Erkenntnis, wurde der durch die Implementation entstehende Aufwand und das Risiko eines Angriffes abgeschätzt. Zusammen mit der Betreuungsperson wurde anschliessend entschieden, die Sicherung des Tokens nicht vorzunehmen, da der Aufwand, im Vergleich zum dadurch verminderten Risiko, zu gross wäre.

18.1.9.2 Sicherheit Firebase-API-Token

Eine weitere Herausforderung beim Bereitstellen der Web-Applikation war, dass diese mit Firebase kommunizieren muss. Hierzu wird von Firebase ein API-Schlüssel zur Verfügung gestellt, welcher dann für die Verbindung mit Firebase verwendet werden kann. Im Web wird dieser Schlüssel in der Datei `firebase-config.js` abgelegt und mit der Applikation an den Aufrufer ausgeliefert. Das Problem hierbei ist, dass jeder Aufrufende, der die Datei kennt, den API-Schlüssel im Klartext auslesen und somit potenziell auf Firebase zugreifen kann.

Um dieser Sicherheitslücke vorzubeugen, wurden zwei API-Schlüssel generiert. Einer welcher in der Entwicklung verwendet wird und keine Einschränkungen besitzt und ein Zweiter, welcher in der Produktion eingesetzt werden kann. Der Produktions-Schlüssel wird auf beiden Web-Applikationen sowohl `Staging` als auch `Production` verwendet. Dieser Schlüssel hat die Einschränkung, dass er nur Anfragen von diesen beiden Domains zulässt. Dadurch kann sich eine böswillige Person den API-Schlüssel immer noch kopieren, diesen aber nicht verwenden.

18.1.10 Service Locator

Um Services innerhalb der Frontend-Applikation zu erreichen wird mit dem Service Locator gearbeitet. Diese Komponente ist zuständig für das Verwalten und Lokalisieren von Services. Ein Service kann hierbei laut Dokumentation mit dem Aufruf `locator<ServiceName>()` abgerufen werden. Diese Methode verhielt sich allerdings nicht wie erwartet und gab teilweise keinen Service zurück. Dieses Verhalten konnte verhindert werden, indem die `get`-Methode auf dem Service Locator verwendet wurde, anstatt die in der Dokumentation angegebene Methode. Somit musste der Aufruf nach der Korrektur über `locator.get<ServiceName>()` erfolgen [29].

18.1.11 Bilder im Web

Bei der Frontend-Applikation im Web ist, nach Implementierung der ersten Bilder, aufgefallen, dass diese auf Linux Geräten teilweise unscharf dargestellt werden. Dies ist allerdings nicht abhängig von der Bildqualität. Es spielt ausserdem keine Rolle, ob Chromium oder Firefox als Browser verwendet wird. Die Vermutung lag daher nahe, dass der Rendering-Mechanismus von Flutter die Ursache dafür sein könnte. Leider konnte auch durch das Verwenden anderer, von Flutter angebotenen, Renderer kein besseres Ergebnis erzielt werden. Da das Problem allerdings nur bei Linux auftritt und die Zielgruppe

nur Windows verwendet wurde, darauf verzichtet, noch weitere Ressourcen in die Lösung dieses Problems zu investieren.

Abbildung 18.2: Unscharfe Kanten beim Logo der Pronto AG

18.2 Backend

In diesem Kapitel werden die Herausforderungen beschrieben, welche bei der Implementation des Backend aufgetreten sind. Es wurde jene Herausforderungen dokumentiert, welche während der Implementation besonders viel Aufmerksamkeit benötigten.

18.2.1 Autorisierung

Ein wichtiger Bestandteil der Applikation ist die Authentifizierung und Autorisierung von Benutzern. Während die Authentifizierung mit moderatem Aufwand über ASP.NET Bordmittel implementiert werden konnte, war die Autorisierung ein grösseres Unterfangen. Die Authentifizierung wird mithilfe von ASP.NET und einem JWT-Token abgewickelt. Sobald diese aber abgeschlossen ist, muss entschieden werden ob der authentifzierte Benutzer auch die Autorisierung hat, eine Operation durchzuführen.

Hierzu wurden einige `Policies` definiert welche im Autorisierungs-Attribut spezifiziert werden können.

```
1 [Authorize(Policy = "ViewUser")]
2 public IQueryable<User> Users(){...}
```

Auflistung 18.12: Policy auf Operation

Diese arbeiten, mit dem im Rahmen dieser Arbeit geschriebenen `DepartmentAccessAuthorizationHandler`, um zu erkennen, ob ein Benutzer die benötigten Rechte besitzt um die `Policy` zu erfüllen und somit die Operation auszuführen. Hierbei wird die Berechtigungsliste des Benutzers analysiert und die vorhandenen Berechtigungen mit denjenigen welche die `Policy` benötigt abgeglichen.

Die grösste Schwierigkeit hierbei war, das Abbilden der Abteilungs-spezifischen Berechtigungen. Es musste erkannt werden können, für welche Abteilung eine Operation abgesetzt wurde. Hierzu wurde ein neues Attribut erstellt. Das `AccessObjectIdArgument`-Attribut hilft der Autorisierungs-Logik zu erkennen, welche Abteilung von einer Operation betroffen ist. Hierbei wird mit Reflection gearbeitet um die Abteilung herauszufinden. Es werden in dieser Logik vier Varianten unterschieden:

1. Keine Abteilung

Wenn die Berechtigung einer Operation nur global definiert ist und keiner Abteilung zugewiesen werden kann, so wird das `AccessObjectIdArgument`-Attribut nicht gesetzt. Dies bedeutet, dass nur Benutzer, welche globale Berechtigungen in der `Policy` besitzen, zugelassen werden. Diejenigen mit Abteilungs-spezifischen Berechtigungen werden abgewiesen.

```
[Authorize(Policy = "EditDepartment")]
public async Task<Department> CreateDepartment(
  string name){...}
```

Auflistung 18.13: Operation mit globaler Gültigkeit

2. Abteilung nicht relevant

Dieser Fall tritt bei Operationen ein, bei welchen die Abteilung keine Rolle spielt oder weitere Überprüfungen in der Operations-Logik selbst geschehen. In diesem Fall wird das `AccessObjectIdArgument`-Attribut auf „IGNORED“ gesetzt. Somit werden Benutzer mit Globalen Berechtigungen, sowie auch Benutzer mit Abteilungs-spezifischen Berechtigungen zugelassen unabhängig davon, auf welche Abteilungen Sie Berechtigungen haben.

```
[Authorize(Policy = "ViewUser")]
[AccessObjectIdArgument("IGNORED")]
public IQueryable<User> Users(){...}
```

Auflistung 18.14: Operation ohne Abteilungs-Einschränkung

3. Abteilung in Operation

Wenn in einer Operation die Id der Abteilung direkt angegeben werden muss, so tritt dieser Fall ein. Hierbei wird das zweite Argument des `AccessObjectIdArgument`-Attribut auf `true` gesetzt. Dies teilt der Autorisierungs-Logik mit, dass es sich beim angegebenen Namen um ein Argument handelt, welches die Abteilungs-Id beinhaltet. Die Autorisierungs-Logik überprüft anschliessend, ob der Benutzer globale Rechte hat und autorisiert ihn falls dies der Fall ist. Sollte das nicht zutreffen, der Benutzer aber Abteilungs-spezifische Rechte haben, so wird die Abteilung des Benutzers mit derjenigen im angegebenen Argument verglichen und der Benutzer bei einer Übereinstimmung autorisiert.

```
[Authorize(Policy = "EditUser")]
[AccessObjectIdArgument("departmentId", true)]
public async Task<User> CreateUser(
  string userName,
  string password,
  AccessControlList accessControlList,
  int departmentId){...}
```

Auflistung 18.15: Operation mit integrierter Abteilungs-Id

4. Abteilung indirekt in Operation

Dies ist der häufigste und auch der komplexeste Fall. Selbst wenn in der Operation selbst die Abteilungs-Id nicht angegeben ist, so kann diese im Objekt, welches von der Operati-

on betroffen ist, doch vorhanden sein. In diesem Fall wird im `AccessObjectIdArgument`-Attribut das Argument mit der Id des betroffenen Objektes angegeben und der zweite Parameter auf `false` gesetzt (ist Standardmässig `false`). Hierbei ist wichtig zu beachten, dass das betroffenen Objekt das Interface `IDepartmentComparable` implementiert haben muss, womit die Autorisierungs-Logik in der Lage ist, die Abteilung des Objektes zu analysieren. Die Autorisierungs-Logik wird anschliessend prüfen, ob der Benutzer globale Rechte hat und ihn gegebenenfalls autorisieren. Falls dies nicht der Fall ist, der Benutzer aber Abteilungs-spezifische Rechte hat, wird die Abteilung des Benutzers mit derjenigen im angegebenen Objekt verglichen und der Benutzer bei einer Übereinstimmung autorisiert.

```
1 [Authorize(Policy = "EditDeploymentPlan")]
2 [AccessObjectIdArgument("id")]
3 public async Task<int> RemoveDeploymentPlan(
4   int id){...}
```

Auflistung 18.16: Operation ohne integrierte Abteilungs-Id

In der Auflistung 18.16 wird die Abteilungs-Id des Einsatzplans, mit derjenigen des Benutzers verglichen. Das Einsatzplan-Objekt muss hierfür allerdings `IDepartmentComparable` implementieren.

18.2.2 GraphQL Logging

Um im Fehlerfall eine gute Analyse zu gewährleisten und die Chance auf eine erfolgreiche Fehlerbehebung zu erhöhen, ist Logging ein wichtiges Mittel. HotChocolate bietet von sich aus kein Logging von GraphQL-Anfragen an. Glücklicherweise konnte, mithilfe einer Anleitung, das Logging von GraphQL-Anfragen implementiert werden [32]. Nachdem das Logging implementiert war, kamen allerdings zwei weitere Problemstellungen zum Vorschein, welche behandelt werden mussten:

- Log Correlation
- Sensitive Daten

18.2.2.1 Log Correlation

Bei der „Log Correlation “ geht es darum, Fehler welche auf dem Frontend passieren im Backend identifizieren zu können. Die einfachste Art ist hierbei die Zeit des Fehlerauftritts im Frontend mit dem Log im Backend abzugleichen und so den Fehler aufzuspüren. Besonders bei einer API, bei welcher potenziell sehr viele Anfragen gleichzeitig ankommen, ist diese Methode allerdings nicht praktikabel. Dies da Log-Einträge im Backend, aufgrund der geringen Zeitabstände verschiedener Anfragen, nicht eindeutig dem auftretenden Fehler im Frontend zugewiesen werden können.

Dieses Problem wurde mit dem Einsatz einer eindeutigen Id gelöst, welche für jede Anfrage generiert wird. Diese wird dem Frontend im Fehlerfall mitgegeben und auch im Backend ins Log geschrieben. Somit kann jeder Fehler im Frontend eindeutig im Log des Backend identifiziert werden. Zur Generierung der sogenannten „TraceId“ wurde die, in ASP.NET bereits integrierte Lösung verwendet, welche jeder Anfrage eine eindeutige Id zuweist. Es musste Log Scoping aktiviert werden, damit diese Id auch im Log dargestellt wird [33]. Anschliessend wurde die GraphQL-API so konfiguriert, dass die Id dem Frontend im Fehlerfall angezeigt wird.

Abbildung 18.3: TraceId bei fehlgeschlagener GraphQL-Anfrage

18.2.2.2 Sensitive Daten

Aufgrund des nun eingeführten Logging von Anfragen wurden alle Parameter, Variablen und Inhalte von GraphQL-Anfragen im Backend ins Log geschrieben. Bereits nach kurzer Zeit fiel auf, dass auch Daten geloggt werden, welche nirgendwo im Klartext ersichtlich sein sollten. Namentlich geht es dabei um Passwörter und andere sensitive Daten, welche als Parameter oder Variablen in Anfragen übergeben werden.

```
Request ended. Content was:
+-+-+-+-+-+-+-+-+-+-+
{
  authenticate(userName: "Admin", password:"ProntoMIA.")
}
+-+-+-+-+-+-+-+-+-+-+
Time needed: 189 milliseconds.
```

Auflistung 18.17: Log enthält sensitiven Daten

Um dieses Logging von sensitiven Informationen zu verhindern, musste der Logger aus der Anleitung angepasst werden. Erst mussten allerdings die sensitiven Informationen in einer Operation erkannt werden können. Hierzu wurde ein neues Attribut namens „Sensitive“ erstellt. Dieses kann bei Operationen angegeben werden und bestimmt welche Parameter resp. Variablen der Operation sensitive Daten beinhalten.

```
[Sensitive("password")]
public async Task<string> Authenticate(
  string userName,
  string password){...}
```

Auflistung 18.18: Attribut bei Operation

Der Logger liest anschliessend alle sensitiven Attribute der Applikation aus und vergleicht deren Inhalt mit den Parametern und Variablen der eingehenden Anfragen. Falls eine Übereinstimmung stattfindet, wird der entsprechende Parameter oder dessen Variable im Log unkenntlich gemacht.

```
Request ended. Content was:
+-+-+-+-+-+-+-+-+-+-+
{
  authenticate(userName: "Admin", password:"***")
}
+-+-+-+-+-+-+-+-+-+-+
Time needed: 10 milliseconds.
```

Auflistung 18.19: Log mit sensitivem Parameter

```
Request ended. Content was:
+-+-+-+-+-+-+-+-+-+-+
query($pw: String) {
  authenticate(userName: "Admin", password: $pw)
}

+-+-+-+-+-+-+-+-+-+-+
Variables:
pw : *** : HotChocolate.Types.StringType
+-+-+-+-+-+-+-+-+-+-+
Time needed: 20 milliseconds.
```

Auflistung 18.20: Log mit sensitiver Variable

Eine noch bestehende Einschränkung des Loggers ist, dass die sensitiven Parameter global gesammelt werden. Dies bedeutet, dass ein in einer Operation als sensitive gekennzeichneter Parametername auch in jeder anderen Operation als sensitiv angesehen wird. Da dieses Verhalten für diese Arbeit nicht hinderlich ist und eine Änderung einen grossen Mehraufwand bedeutet hätte, wurde entschieden dies so zu belassen.

18.2.3 Firebase und Testing

In der Backend-Komponente wird Firebase verwendet, um Push-Benachrichtigungen an die Geräte von Benutzern der Applikation zu senden. Hierzu wird die von Firebase zur Verfügung gestellte Programmierschnittstelle für C# genutzt [34].

Beim Testing fiel auf, dass alle Klassen dieser Schnittstelle welche vom Server verwendet werden als `sealed`, `protected` oder `private` deklariert sind. Dieser Umstand verunmöglicht ein Mocking der Klassen und somit das Testen von Funktionen, welche Methoden dieser Klassen verwenden. Da ein wichtiger Teil der Funktionalität der Applikation das Senden von Push-Benachrichtigungen ist, war ein Verzicht auf das Testen keine Option. Folglich musste eine Lösung für das Problem gefunden werden.

Um die Einschränkung zu umgehen, wurde ein Adapter `FirebaseMessagingAdapter` und ein dazugehöriges Interface `IFirebaseMessagingAdapter` erstellt. Der Adapter implementiert alle vom Backend verwendeten Methoden der Programmierschnittstelle und leitet die Aufrufe an diese weiter. Dadurch, dass das Backend die Programmierschnittstelle nun nur

noch über das Interface des Adapter aufruft und dieses gemockt werden kann, ist ein Testing der aufrufenden Methoden ohne Problem möglich.

18.2.4 Firebase Token Verwaltung

Jeder Benutzer der Applikation kann potenziell von mehreren Geräten auf diese zugreifen. Dies resultiert in mehreren Firebase-Tokens, welche es ermöglichen den Benutzer auf all seinen Geräten zu erreichen. Falls ein Gerät lange oder gar nicht mehr benutzt wird, verliert der entsprechende Firebase-Token seine Gültigkeit. Um die Datenbank nicht unkontrolliert anwachsen zu lassen, muss dementsprechend sichergestellt werden, dass abgelaufene Token aus dieser entfernt werden.

Daher wurde eine Funktion geschrieben, welche nach einer Sende-Operation die Antwort von Firebase analysiert, erkennt welche Token abgelaufen sind und diese entfernt. Leider ist es nur möglich, zu erkennen, ob ein Token abgelaufen ist, indem man versucht eine Nachricht an diesen zu senden. Dies kann allerdings auch eine als Test markierte Nachricht sein, welche einen Sendevorgang lediglich simuliert. Um nicht unnötige Anfragen an Firebase machen zu müssen, wurde die Logik so implementiert, dass diese nach jedem Senden einer Push-Benachrichtigung die Antwort analysiert und die abgelaufenen Token löscht. Es wurde hierbei in Kauf genommen, dass die Sende-Operation aufgrund des Aufräumens etwas länger dauert. Die Alternative wäre ein regelmässiger Hintergrundjob gewesen, welcher alle Tokens überprüft und Abgelaufene gelöscht hätte. Der Implementierungsaufwand einer solchen Lösung wäre allerdings grösser, als der Nachteil der gering langsameren Operation. Sollte die Applikation allerdings in Zukunft mit mehreren 100’000 Tokens umgehen müssen, muss diese Option wieder in Erwägung gezogen werden.

19 | Usability Tests

Im Rahmen der Qualitätssicherung dieser Arbeit, war es vorgesehen, Usability-Tests durchzuführen. Diese Tests waren gegen Ende der Construction-Phase (3.2.3) eingeplant. Zu diesem Zeitpunk war allerdings, Aufgrund der vielen Herausforderungen aus Kapitel 18, Version v0.2.0 gerade erst fertig geworden. Da, um einen für den Kunden zufriedenstellenden Projektabschluss zu ermöglichen, mindestens Version v0.3.0 fertiggestellt werden sollte, wurde zusammen mit dem Kunden entschieden, die Usability-Tests zu diesem Zeitpunkt nicht durchzuführen und die verbleibende Zeit in die Fertigstellung von v0.3.0 zu investieren.

Es wird bei der Weiterführung des Projekts allerdings dringend empfohlen, solche Tests durchzuführen. Diese können viel über die Gewohnheiten und Erwartungen der Benutzer aussagen und helfen eine intuitivere Applikation zu gestalten. Dies führt wiederum dazu, dass die Applikation aktiver genutzt wird, was dem Kunden zugute kommt.

20 Qualitätssicherung

In diesem Kapitel werden die, in Kapitel 6 analysierten, Risiken nochmals zur Hand genommen und geprüft, ob und wie diese eingetreten sind. Auch werden die in Kapitel 8.3 definierten Nicht-funktionalen Anforderungen auf ihre Einhaltung während der Umsetzung überprüft.

20.1 Risikomanagement

Im Kapitel 6 wurden die möglichen Risiken und ihre Auswirkungen auf das Projekt analysiert und dokumentiert. In diesem Kapitel soll aufgezeigt werden, welche vorhergesehenen Risiken eingetreten sind und welche unvorhergesehenen dazukamen.

20.1.1 Ausfall eines Teammitgliedes

Das vorhergesehene Risiko „Ausfall eines Teammitgliedes“ aus Kapitel 6.1.1 ist während dieser Arbeit zweimal eingetreten. Hierbei fiel eines der beiden Teammitglieder für jeweils zwei Arbeitstage aus. Dies bedeutete einen Schaden von 32h was glücklicherweise noch unter dem geschätzten maximalen Schaden von 36h für dieses Risiko liegt.

Durch die gute Absprache innerhalb des Teams konnte das verbleibende Teammitglied ungehindert weiterarbeiten. Durch zusätzliche Stunden am Wochenende und in Zwischenstunden konnte ein Teil der verlorenen Zeit aufgeholt werden.

20.1.2 Flutter Plattform Inkompatibilität

Ein unvorhergesehenes Risiko, welches eingetreten ist, war die Inkompatibilität vieler Dart-Pakete mit der Flutter Web-Plattform. Die Tatsache, dass mit Flutter 2.0 die Plattform „Web“als stabil angegeben wurde, hat das Team von der Verwendung der Technologie überzeugt. Hierbei wurde allerdings nicht beachtet, dass viele Pakete, welche das Flutter-Framework um zusätzliche Funktionalität erweitern, diese Plattform noch nicht oder nur ungenügend unterstützen.

Aufgrund dieses Umstandes musste ein enormer Mehraufwand bei der Implementierung der Web-Plattform betrieben werden. Viele der Inkompatibilitäten in den Paketen waren ausserdem ungenügend oder gar nicht dokumentiert respektive bekannt. Es wurden insgesamt gut 40h in das Lösen solcher Inkompatibilitäts-Probleme investiert.

Trotz dieser Schwierigkeiten wird das Flutter-Framework weiterhin als die richtige Wahl angesehen. Dieses Risiko hätte allerdings schon in der Vorstudie erkannt und eingeplant werden sollen, um dem Kunden einen realistischeren Zeitplan mitteilen zu können. Dank

dem Releaseplan aus Kapitel 8.5 konnte dem Kunden obschon dessen, dass der geplante Release nicht erreicht werden konnte, ein zufriedenstellendes Produkt abgeliefert werden.

20.2 Nicht-funktionale Anforderungen

In diesem Kapitel werden die „Nicht-funktionalen Anforderungen“ aus Kapitel 8.3 analysiert und dokumentiert, ob und wie diese erfüllt wurden.

20.2.1 Funktionalität

In diesem Kapitel sind die NFRs aufgeführt, welche mit der Funktionalität der Applikation zu tun haben.

NFR01

Dieses NFR besagt, dass das Auslesen von Daten eine erfolgreiche Anmeldung benötigt. Dies wurde durch die Implementation einer Autorisierung in der Applikation umgesetzt.

20.2.2 Benutzbarkeit

In diesem Kapitel werden alle NFRs erläutert, welche mit der Benutzbarkeit der Applikation zusammenhängen.

NFR02

In diesem NFR geht es darum, dass die Arbeitsschritte der Aktoren „Mitarbeiter“ und „Reinigungskraft“ selbsterklärend sein sollen. Dies konnte nicht mithilfe von Usability-Tests überprüft werden, jedoch wird aufgrund der Übersichtlichkeit und simplen Gestaltung der Benutzeroberfläche davon ausgegangen, dass diese Anforderung erfüllt werden konnte. Die Einfachheit der Benutzeroberfläche wurde ausserdem vom Kunden bestätigt.

NFR03

Dieses NFR besagt, dass jegliche Information welche farbcodiert ist auch auf andere Art visualisiert werden muss, um farbenblinde Personen zu unterstützen. Es gibt in der Applikation, wie von diesem NFR verlangt, keine Information welche nur mithilfe der Farbe visualisiert wird.

NFR04

Dieses NFR befasst sich mit der textuellen Repräsentation innerhalb der Applikation. Da nicht alle Benutzer gut Deutsch können, sollen die Informationen nicht nur textuell dargestellt werden. Dies ist grundsätzlich gelungen. In der Administration von Benutzern und Einsatzplänen werden allerdings einige Informationen nur textuell dargestellt und diese Aktionen erfordern daher eine gewisse Deutsch-Kenntnis.

20.2.3 Zuverlässigkeit

Dieser Abschnitt befasst sich mit NFRs, welche die Zuverlässigkeit der Applikation sicherstellen sollen.

NFR05

In diesem NFR wird die Fehlerbehandlung innerhalb der App thematisiert. Diese soll dem Benutzer sprechende Fehlermeldungen anzeigen und global implementiert sein. Da sowohl im Backend, wie auch im Frontend eine globale Fehlerbehandlung implementiert ist, wurde dieses NFR erfüllt.

20.2.4 Änderbarkeit

In diesem Kapitel geht es um NFRs, welche sich mit der Änderbarkeit respektive Wartbarkeit der Applikation beschäftigen. Da es sich hierbei um Fragen der Code-Qualität handelt wird pro NFR jeweils zusätzlich zwischen Frontend und Backend unterschieden.

NFR06

Dieses NFR besagt, dass Klassen innerhalb der Applikation nicht länger als 300 Zeilen lang sein dürfen, um die Lesbarkeit von Klassen zu garantieren.

Frontend:
Im Frontend wurde dieses NFR manuell geprüft, da kein Tool gefunden werden konnte, um dies zu automatisieren. Der Test-Code wurde bei dieser Analyse ignoriert. Bei keiner der überprüften Klassen konnte eine höhere Anzahl Zeilen als 274 festgestellt werden. Bei der Klasse mit 274 Zeilen handelt es sich um `UserEditView`, welche UI-Elemente für das Benutzerformular generiert. Sollte das Formular weiter wachsen, muss die Auslagerung einiger UI-Elemente in Betracht gezogen werden.

Backend:
Dieses NFR wurde im Backend mithilfe eines Linter überprüft und weitestgehend eingehalten. Wo es nicht eingehalten werden konnte, wurde dies mit einem Kommentar signalisiert und begründet. Zumeist wurden Klassen mit starker Kohäsion durch Dokumentations-Kommentare aufgeblasen und so grösser als 300 Zeilen. In solchen Fällen wurde dann eine Ausnahme definiert.

NFR07

Dieses NFR besagt, dass Methoden nicht länger als 30 Zeilen lang sein dürfen, um die Lesbarkeit von Methoden zu garantieren.

Frontend:
Im Frontend wurde dieses NFR per Linter geprüft. In fünf Fällen konnte die Regel nicht eingehalten werden. Bei vier davon handelt es sich um Methoden, die UI-Elemente generieren. Diese lassen sich nicht sinnvoll aufteilen, da der Grund für die erhöhte Anzahl an Zeilen auf die vielen Optionen zurückzuführen ist, die einige der verwendeten Elemente benötigen. Beim letzten Fall, bei welchem die Regel nicht eingehalten werden konnte, handelt es sich um die Methode `UserEditViewModel.modifyAccessControlList`. Diese

Methode beherbergt ein Switch-Case-Statement, welches sich aufgrund der vielen verschiedenen Fälle beim Ändern der AccessControlList nicht trennen lässt. Es wurde daher entschieden, alle erwähnten Methoden so zu belassen.

Backend:
Diese Anforderung wurde im Backend mithilfe eines Linter geprüft und eingehalten. Vereinzelt wurden bei starker Kohäsion innerhalb der Methode oder einfachen Konstrukten wie `Switch`, welche lange Methoden zur Folge haben, Ausnahmen definiert und dokumentiert.

NFR08

Dieses NFR besagt, dass Zeilen nicht länger als 80 Zeichen lang sein dürfen.

Frontend:
Im Frontend wurde dieses NFR per Linter geprüft und konnte über den gesamten Code eingehalten werden. Zusätzlich wird der Code aber auch noch durch das Flutter-eigene Format-Tool in das Einhalten der Regel gezwungen, da dieses den Code entsprechend formatiert.

Backend:
Dieses NFR wurde im Backend mithilfe eines Linter geprüft und eingehalten. Es wurden wenige Ausnahmen, primär im Test-Code definiert, um lange `Strings` auf einer Linie platzieren zu können.

NFR09

Dieses NFR besagt, dass der McCabe-Wert jeder Methode unter 10 liegen muss.

Frontend:
Im Frontend wurde dieses NFR per Linter geprüft. In vier Fällen konnte die Regel nicht eingehalten werden. Bei drei davon handelt es sich um Methoden, deren alleinige Aufgabe es ist, über ein Switch-Case-Statement zwischen mehreren Fällen zu unterscheiden. Hierbei ist es nicht möglich, zu einer Einhaltung der Regel zu kommen, da der Switch-Case nicht aufgeteilt werden soll/kann. Beim letzten Fall handelt es sich um die Methode `AccessControlList.isEqual`, welche den Wertevergleich zweier `AccessControlList`-Objekte ermöglicht. Da Dart unglücklicherweise keine Alternative zum schrittweisen Vergleich der elf Attribute bietet, ist ein Refactoring hier nicht möglich. Es wurde daher entschieden, alle erwähnten Methoden so zu belassen.

Backend:
Der McCabe-Wert der Methoden im Backend wurde mithilfe einer Erweiterung in der IDE überprüft und analysiert. Bis auf sechs Methoden ist der Wert sogar unter oder gleich fünf. Von diesen sechs Methoden, welche einen McCabe-Wert über fünf haben, sind lediglich zwei über dem vorgegebenen Limit von zehn.

Dies sind die `HasControl`-Methode in der Klasse `AccessControlListExtension` (McCabe-Wert 13) und die `Message`-Methode in der Klasse `ErrorExtensions` (McCabe-Wert 15). Da beide Methoden nur triviale Logik beinhalten,werden keine weiteren Massnahmen bezüglich dieser Methoden ergriffen. Der McCabe-Wert wurde in beiden Fällen aufgrund des enthaltenen `Switch`-Statement in die Höhe getrieben.

NFR10

Dieses NFR besagt, dass Methoden innerhalb der Applikation höchstens fünf Argumente haben dürfen.

Frontend:
Im Frontend wurde dieses NFR per Linter geprüft. In einem Fall konnte die Regel nicht eingehalten werden. Es handelt sich dabei um die Methode `DeploymentPlanService.updateDeploymentPlan`. Man könnte theoretisch als einzelnes Argument ein `DeploymentPlan`-Objekt verlangen, jedoch müsste dies innerhalb der Methode, aufgrund der darin enthaltenen GraphQL-Query direkt wieder auseinandergenommen werden. Aufgrund der Tatsache, dass die Limite nur um ein Argument überschritten wird, wird ein Refactoring als nicht sinnvoll errachtet.

Backend:
Im Backend konnte dieses NFR bei fünf Methoden nicht eingehalten werden. Hierbei handelt es sich bei allen fünf Methoden um solche, welche direkt von der GraphQL-API aufgerufen werden. Ausserdem arbeiten alle von ihnen intern mit mehreren Managern um ihre Aufgabe zu erfüllen. Die Manager selbst erfüllen alle dieses NFR.

Da es sich bei den betroffenen Methoden um API-Methoden handelt, wird die Argumentliste zusätzlich durch die Dependency Injection aufgebläht. Aufgrund dieses Umstandes und aufgrund der Tatsache, dass alle übergebenen Argumente für einen erfolgreichen API-Call gebraucht werden, wurde beschlossen, dass diese Methoden so bestehen bleiben können.

NFR11

Dieses NFR besagt, dass die Testabdeckung des Codes mindestens 80% betragen muss.

Frontend:
Im Frontend beträgt die Testabdeckung des testbaren, nicht automatisch generierten Codes rund 71%. Der gewünschte Wert wurde damit unglücklicherweise nicht erreicht, dennoch konnten die wichtigsten Funktionalitäten, namentlich die Business Logik grössenteils getestet werden. Somit kann eine ausreichende Testabdeckung gewährleistet werden. Die Gründe für die tiefere Abdeckung in einigen Klassen werden nachfolgend erläutert.

Einerseits bietet Flutter keine Möglichkeit, Konstanten wie `kReleaseMode` und `kIsWeb` in einer Testumgebung zu simulieren und anzupassen. Dadurch können einige Unteschеidungen zwischen der Web Plattform und den Mobile Plattformen in der Logik nicht getestet werden.

Des Weiteren lassen sich einige Methoden und daruch die zugehörigen Klassen nicht von der verwendeten Mock-Library `Mockito` mocken. Es handelt sich dabei konkret um Methoden mit generischen Rückgabewerten, welche nicht `null` sein können. In der neuesten Version von Mockito gibt es dazu eine Lösung, jedoch kann diese Version aufgrund von Konflikten mit anderen Libraries momentan nicht installiert werden [35]. Mockito 5.0.0 hängt nämlich von HTTP 0.13.0 ab, wobei andere Libraries sich noch auf HTTP 0.12.0 befinden.

Aufgrund dieser Umstände ist das Entwicklerteam der Meinung, dass eine Abdeckung von 71% ausreichend ist.

Backend:
Im Backend beträgt die Testabdeckung des testbaren, nicht automatisch generierten Codes 91%. Somit wurde der gewünschte Wert erreicht und sogar übertroffen.

NFR12

Dieses NFR besagt, dass alle Aufrufszenarien der API dokumentiert werden müssen.

Im Backend wurde für jeden möglichen API-Aufruf ein Dokumentations-Kommentar angelegt, welcher den Aufruf und dessen Argumente beschreibt.

NFR13

Dieses NFR besagt, dass eine Installations- sowie eine Wartungsanleitung erstellt werden soll.

Es wurde sowohl im Frontend als auch im Backend eine Installationsanleitung in der Form einer README-Datei angelegt:

- Frontend:
  - https://github.com/Pronto-AG/Pronto-MIA-App/blob/master/README.md
- Backend:
  - https://github.com/Pronto-AG/Pronto-MIA-Server/blob/master/README.md

Auf das Erstellen einer Wartungsanleitung wurde verzichtet, da diese vom Kunden nicht gewünscht war.

21 Zeitauswertung

In diesem Abschnitt wird der Zeitplan ausgewertet. Dies beinhaltet eine Analyse des Zeitaufwandes und die Evaluation der Einhaltung der einzelnen Meilensteine.

21.1 Meilensteine

Nachfolgend wird die Einhaltung der Meilensteine geprüft und eventuelle Eigenheiten der einzelnen Arbeitsprodukte erläutert. Hier gilt zu beachten, dass als Referenz zur Fertigstellung des Meilensteins nicht der Abschluss des Meilensteins in GitHub genommen wurde. Es wurde sich vorwiegend and den Beschlüssen aus den Sitzungsprotokollen orientiert.

21.1.1 Projektplan

Der Meilenstein Projektplan wurde auf den **07.03.2021** geplant und konnte etwas früher am **05.03.2021** fertiggestellt werden. Da die Erstellung eines Projektplans bereits aus der Studienarbeit und dem Engineering-Projekt bekannt war, konnte dieser zügig erstellt und bereits eine Woche vorher mit dem Betreuer besprochen werden. Die Abnahme durch den Kunden erfolgte jedoch erst am 05.03.2021.

21.1.2 Anforderungsanalyse

Der Meilenstein Anforderungsanalyse wurde auf den **21.03.2021** geplant und konnte termingerecht am **21.03.2021** abgeschlossen werden. Die letzten Unklarheiten der Anforderungsanalyse konnten an der Sitzung vom 18.03.2021 mit dem Kunden besprochen und geklärt werden. Auf eine Schnittstellenbeschreibung wurde aufgrund der gewählten API-Technologie vorerst verzichtet, da sich das Projektteam erst in GraphQL einarbeiten musste.

21.1.3 End of Elaboration

Der Meilenstein End of Elaboration wurde auf den **04.04.2021** geplant und konnte termingerecht am **04.04.2021** abgeschlossen werden. Die End of Elaboration Checklist wurde lediglich im Projektteam besprochen. Der Prototyp hingegen, welcher auch als Grundlage der Architektur dient, wurde dem Kunden vorgeführt.

21.1.4 Qualitätsmassnahmen

Der Meilenstein Qualitätsmassnahmen wurde auf den **16.05.2021** geplant, konnte jedoch erst einige Tage später, am **20.05.2021** mit der Fertigstellung von v0.2.0 abgeschlossen werden. Dies wurde jedoch zum ursprünglichen Termin mit dem Kunden besprochen und von diesem abgesegnet.

21.1.5 End of Construction

Der Meilenstein End of Construction wurde auf den **30.05.2021** geplant, konnte jedoch erst eine Woche später, am **06.06.2021** abgeschlossen werden. Aufgrund der Verzögerungen während der Entwicklung hat man sich, in Absprache mit dem Kunden, entschieden die Construction-Phase um eine Woche zu verlängern um v0.3.0 komplett abschliessen zu können. Diese Entscheidung wurde gefällt, um dem Kunden zum Ende dieser Arbeit eine Applikation mit vollständig implementierter Einsatzplanverwaltung bieten zu können.

21.1.6 End of Documentation

Der Meilenstein End of Documentation wurde auf den **13.06.2021** geplant, konnte jedoch erst am **14.06.2021** abgeschlossen werden. Dies ist auf die Verlängerung der Construction Phase im Meilenstein End of Construction zurückzuführen.

21.1.7 Abgabe

Der Meilenstein Abgabe, wurde auf den **18.06.2021** geplant und wird termingerecht am **18.06.2021** abgeschlossen werden. Trotz der Verzögerungen, während der Construction-Phase, konnte die Dokumentation zügig fertiggestellt werden, was eine termingerechte Abgabe ermöglicht.

21.2 Zeitrapport

Gemäss Kapitel 2.3 beschränkt sich der Zeitaufwand der Arbeit auf gesamthaft 720h, wobei je 360h pro Mitglied des Projektteams zu leisten sind. Der Zeitplan dieser Arbeit wurde unter Berücksichtigung dieser 720h erstellt.

Der tatsächlich erreichte Zeitaufwand beträgt **739h 51min**, gemäss dem Stand vom 17.06.2021. Nachfolgend wird dieser Zeitaufwand in einigen Grafiken ausgewertet und genauer beleuchtet.

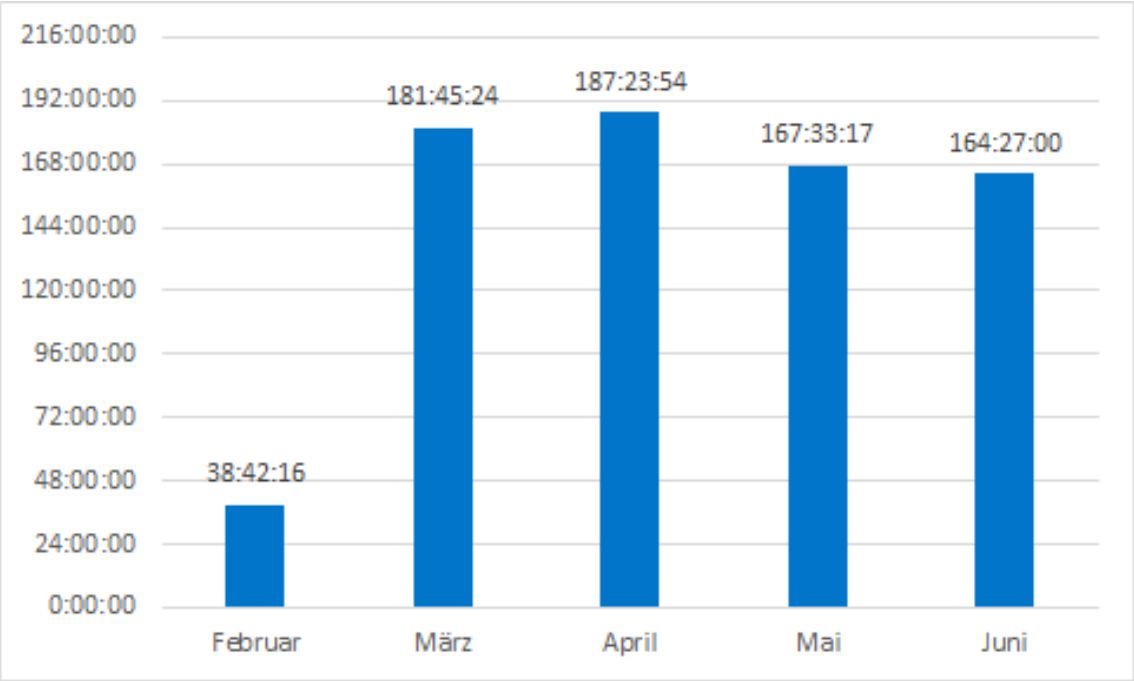

Abbildung 21.1: Zeitrapport, ausgewertet nach Monat

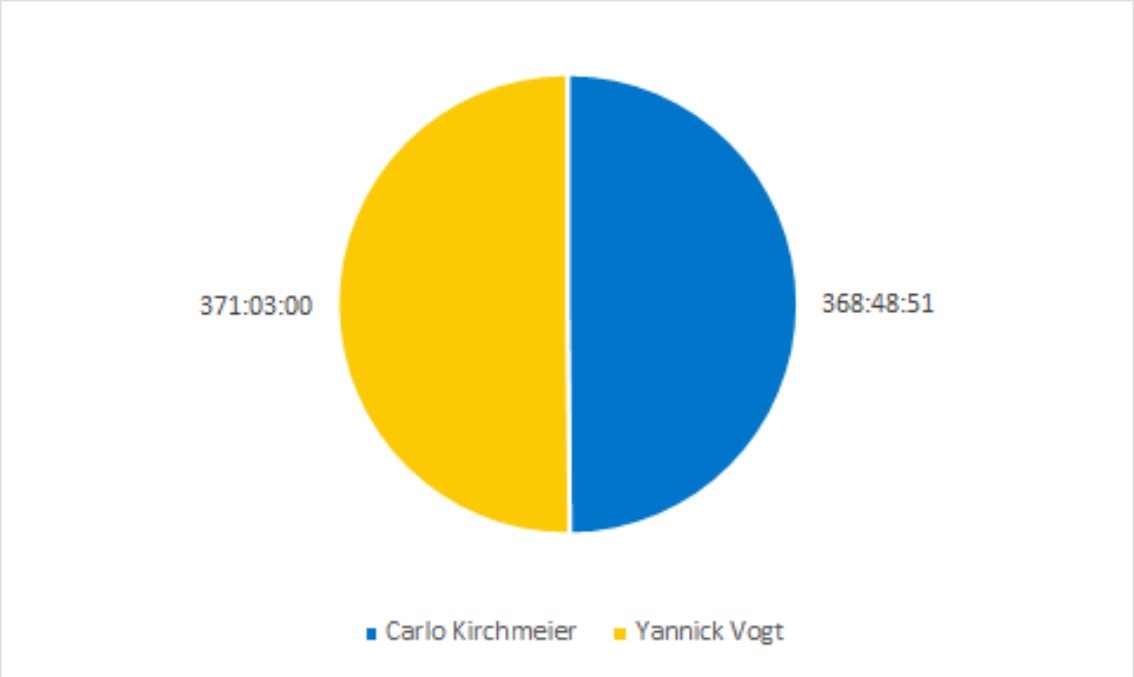

Abbildung 21.2: Zeitrapport, ausgewertet nach Personen

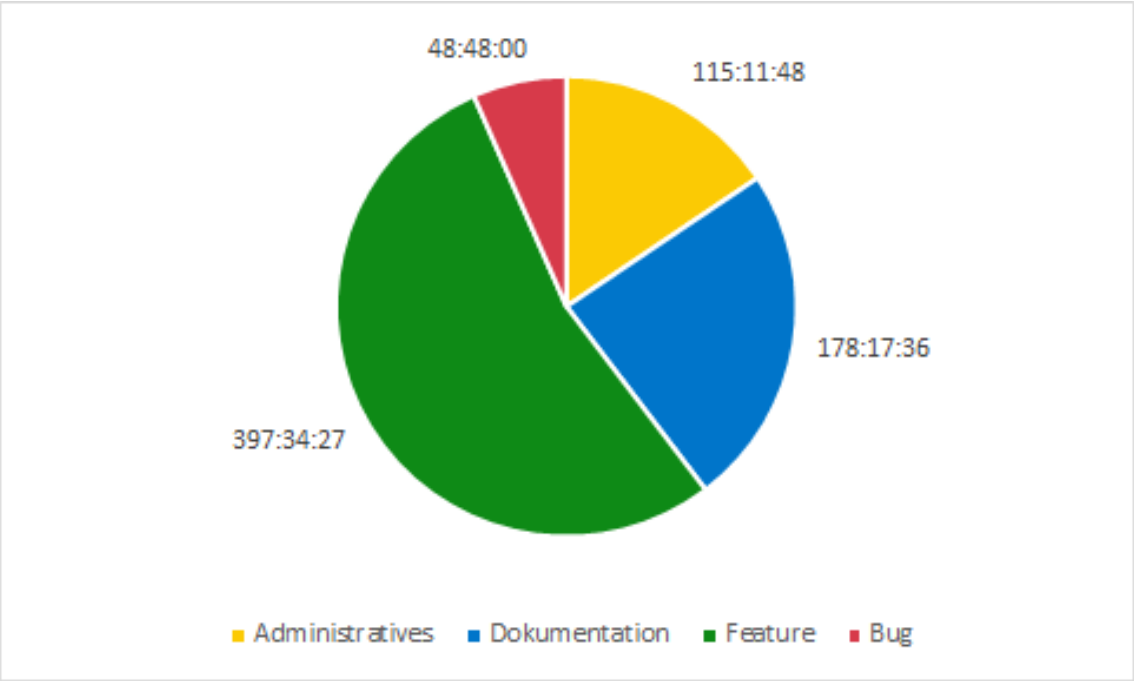

Abbildung 21.3: Zeitrapport, ausgewertet nach Kategorie

22 Schlussfolgerungen

In diesem Abschnitt wird ein Fazit über die gesamte Arbeit gezogen. Ausserdem wird ein Ausblick in die Zukunft unternommen und erläutert, wie es voraussichtlich nach dieser Arbeit mit der Applikation und dem Projekt weitergehen wird.

22.1 Fazit

Das Frontend und das Backend, welche gemeinsam die Applikation „Pronto MIA“ darstellen und im Rahmen dieser Arbeit entwickelt wurden, erfüllen die wichtigsten Bedürfnisse des Kunden. Zusätzlich bieten sie eine solide Basis für eine nachhaltige Weiterentwicklung der Applikation.

Obschon nicht alle Wünsche des Kunden bereits in dieser Arbeit umgesetzt wurden, konnten doch die Einsatzplan-Verwaltung, die Benutzer-Verwaltung und die Benachrichtigung der Mitarbeitenden korrekt und solide implementiert werden. Auch war von Beginn weg klar, dass nicht alle definierten Szenarien im Rahmen dieser Arbeit implementiert werden könnten. Die letzte Version, welche erreicht werden konnte, ist v0.3.0.

Besonders zu beachten gilt, dass bei der Implementierung auf eine gute Wartungsfähigkeit, Testabdeckung und Erweiterbarkeit geachtet wurde. Somit wurde der Grundstein für eine erfolgreiche Weiterführung des Projektes gelegt. Erwähnenswert ist ausserdem, dass die Applikation mittlerweile im Play-Store und auf dem Web verfügbar ist. Ausserdem ist die App im „TestFlight“ von Apple verfügbar.

Aufgrund dessen, dass die App nur für Mitarbeiter der Pronto AG nutzbar ist, hat Apple deren Veröffentlichung im regulären App-Store nicht zugestimmt. Apple schlägt vor, die App mithilfe des Apple Business Managers an die Geräte der Firma zu verteilen. Diese Verteilung ist allerdings nicht mehr im Umfang dieser Arbeit machbar und wird deshalb auf eine allfällige Folgearbeit vertagt.

Ort	**Link**
Google Play-Store	`https://play.google.com/store/apps/details?id=ch.prontoag.mia`
Web App	`https://app.mia.pronto-ag.ch`

22.2 Ausblick

v0.3.0 wurde im Rahmen dieser Arbeit auf zwei von drei Plattformen komplett ausgerollt und auf einer teilweise. Nach Abschluss dieser Arbeit wird das Projekt voraussichtlich von einem neuen Team, im Rahmen einer Semesterarbeit, übernommen werden. Aufgrund der guten Dokumentation und Testabdeckung sollte diese Übernahme ohne grössere Probleme stattfinden können.

Das neue Team wird sich sicher mit dem Thema Apple Business Manager auseinandersetzten müssen, um die App auch im App Store von Apple anbieten zu können. Grundsätzlich kann das neue Team den Releasplan dieser Arbeit 1 zu 1 übernehmen, es bietet sich allerdings an, zumindest Teile der Anforderungsanalyse nochmals durchzuführen um allfällige Änderungen in die Anforderungen einfliessen zu lassen. Das Team der aktuellen Arbeit wird auch nach Abschluss für das neue Team bei Fragen und allfälligen Unklarheiten zur Verfügung stehen, um sicherzustellen, dass einer Weiterentwicklung der Applikation nichts im Wege steht.

Teil V

Appendix

A Abschlussberichte

A.1 Carlo Kirchmeier

Die Bachelorarbeit ist das bisher grösste Softwareprojekt, welches ich je abschliessen durfte. Auch in Sachen Dokumentation ist dies die bisher umfangreichste, welche ich je geschrieben habe. Diese Arbeit hat viele Ähnlichkeiten mit der Studienarbeit des letzten Semesters, wobei sie allerdings nochmals eine Stufe grösser ist. Durch die zusätzliche Zeit, welche uns als Team zur Verfügung stand, konnte ein grösseres Projekt auf die Beine gestellt werden, welches näher an einem echten Softwareprojekt in der Arbeitswelt ist.

Ein grosser Unterschied zur Studienarbeit aus dem letzten Semester ist ausserdem, dass wir in diesem Projekt mit einem Industriepartner zusammenarbeiten konnten. Einen Kunden zu haben bedeutet, dass die Organisation angepasst werden muss. Ich denke allerdings, wir konnten durch die regelmässigen Besprechungen und die gründliche Anforderungsanalyse, den Kunden gut abholen und mit diesem zusammenarbeiten.

Unglücklich war, dass wir diverse Probleme mit dem Flutter-Framework hatten, welche uns viel Zeit kosteten. Auch andere Arbeitspakete dauerten unerwartet länger als geplant. Daher konnten wir v0.5.0, unser geplantes Ziel nicht erreichen. Aufgrund dessen, dass wir aus genau diesem Grund mit Releases arbeiteten, war die Auswirkung auf die Arbeit allerdings gering. Ich denke, dass unsere Entscheidung, eine solide Lösung mit weniger Funktionalität zu implementieren die richtige war. Dadurch kann die nächste Gruppe auf einer soliden Grundlage loslegen.

Ich konnte in dieser Arbeit viele Erfahrungen sammeln. Durch den Kundenkontakt konnte ich erste Erfahrungen für zukünftige Projekte mit Kunden sammeln. Aber auch im technischen Bereich konnte ich profitieren. ASP.NET kannte ich vorher nur bedingt und konnte durch diese Arbeit wichtige Konzepte davon erlernen. Mit dem Flutter-Framework habe ich zuvor noch nie gearbeitet und konnte nun einen Einblick gewinnen. Auch habe ich diverses über die Eigenheiten der cross-platform-Entwicklung gelernt.

Mit der Arbeit bin ich insgesamt sehr zufrieden. Wir konnten ein zufriedenstellendes Produkt abliefern, welches in der Pronto AG eingesetzt werden kann. Schade finde ich, dass wir dieses innerhalb dieser Arbeit nicht in den Apple App Store bringen konnten. Mit etwas mehr Zeit hätten wir diesen Umstand korrigieren und das Produkt sicherlich um einige Funktionalitäten erweitern können, da die Grundlage dafür nun gelegt ist. Abschliessend denke ich, dass diese Arbeit mir sehr geholfen hat, mich auf zukünftige Informatikprojekte vorzubereiten.

A.2 Yannick Vogt

Wie auch für Carlo ist diese Bachelorarbeit in meiner Laufbahn das bisher grösste Projekt. Des Weiteren ist es auch das erste Mal für mich, eine Software für einen Kunden zu entwickeln. Die Bachelorarbeit zusammen mit einem Kunden zu absolvieren, war auch schon bei der Auswahl der ausgeschriebenen Bachelorarbeiten ein wichtiges Kriterium für mich. Mit der Pronto AG hat sich dieses Kriterium erfüllt. Der enge Austausch mit ihnen war stets angenehm und das erhaltene Feedback hilfreich.

Gerade durch die Freiheit in der Technologiewahl, konnte ich mir aus technischer Sicht viel Neues aneignen. Es hat uns ermöglicht mit aktuellen Technologien zu arbeiten, was meiner Meinung nach für den Lerneffekt sehr von Vorteil ist. Die Arbeit mit Flutter, was für mich den Grossteil meiner Tätigkeit ausmachte, war sehr spannend. Besonders interessant war es, eine Alternative zur Erstellung von Benutzeroberflächen zu HTML und CSS kennenzulernen. Trotz der vielen positiven Aspekte von Flutter stellte uns das Framework allerdings auch vor einige Herausforderungen, welche uns schlussendlich auch Zeit kosteten. Obwohl ich gerne noch mehr Features implementiert hätte, um der Pronto AG ein umfangreicheres Endprodukt bieten zu können, waren gerade diese Herausforderungen am spannendsten für mich.

Neben neuen technischen Erfahrungen konnte ich mich auch anderweitig verbessern. Die gewonnene Kundenerfahrung wird mir sicherlich helfen, mich auf weitere Projekte in der Zukunft vorzubereiten. Auch in Sachen Projektmanagement habe ich bei mir einige Fortschritte feststellen können. Selbst wenn die Studienarbeit in diesem Aspekt sehr ähnlich war, konnte ich mir doch noch neues aneignen und die bestehenden Erfahrungen festigen.

Was mir schon länger bekannt ist, ich aber sicher noch weiter daran arbeiten muss, ist mein Zeitmanagement. Abzuschätzen wie viel Zeit eine Aufgabe benötigt fällt mir schwer. Dies lässt sich in dieser Arbeit sicher auch durch unbekannte Technologien erklären, dennoch möchte ich mich in Zukunft darin verbessern. Dies wird mir dann sicherlich auch im Arbeitsleben weiterhelfen.

Hoffentlich wird das Projekt im nächsten Semester von einer neuen Gruppe weitergeführt werden. Der Anzahl Bewerbungen nach, bin ich in diesem Punkt jedoch zuversichtlich. Ich bin gespannt, wie sich die App weiterentwicklen wird und freue mich schon darauf, Neuigkeiten darüber zu erfahren.

Abschliessend bin ich mit der geleisteten Arbeit und der daraus resultierenden Software überaus zufrieden. Diese Arbeit war für mich eine wertvolle Erfahrung und das daraus gewonnene Wissen wird mich sicherlich auf meinem weiteren Weg begleiten.

B Zusatzinformationen

B.1 Anforderungsanalyse

B.1.1 Fragebogen

B.1.1.1 Allgemein

- Wäre es besser die Administration über eine Webseite abzuwickeln (Verwaltung Einsatzpläne, Benutzerverwaltung, etc.)?
- Stehen uns beim vorhandenen IT-Dienstleister Serverressourcen zur Verfügung?
- Ist eine Benutzeranleitung und eine Betriebsanleitung erwünscht? In welchem Ausmass soll dies erfolgen?

B.1.1.2 Einsatzpläne

- Wer erstellt die Einsatzpläne (Abteilungsleiter / Administration)?
- Müssen alte Einsatzpläne archiviert werden?
- Kann ein Einsatzplan nach der Publikation am gleichen Tag noch geändert werden?
- Beziehen sich sämtliche Arbeitspläne auf jeweils einen Tag?
- Sollen mehrere Einsatzpläne gleichzeitig ersichtlich sein (heute, morgen, übermorgen)?
- Gibt es abteilungsübergreifende Einsatzpläne?
- Die Vorlage des Einsatzplans enthält auch einen Teil zur Rapportierung, soll der Rapport in Zukunft auch ein Teil des Apps sein?
- Welche Felder aus der Vorlage sind relevant für den Einsatzplan?

B.1.1.3 Urlaub

- Wann werden Urlaubstage und freie Tage beantragt (jährlich, individuell)?
- Wie viele Tage müssen jährlich im Voraus beantragt werden?
- Wie lange im Voraus müssen freie Tage mindestens beantragt werden?
- Wer entscheidet ob Urlaubsanträge angenommen oder abgelehnt werden?

B.1.1.4 Schulung

- Sollen neben den Schulungsvideos auch Textanleitung erstellt werden können?
- Sind Bilder für Textanleitungen auch vorgesehen?

- Sollen Schulungsvideos zusätzlich mit Texterklärungen versehen werden können?
- Wer darf Schulungsvideos und Anleitungen publizieren?

B.1.1.5 News

- Wer darf welche Arten von News publizieren (firmenweit, abteilungsspezifisch)?
- Ist die Person, welche den Artikel schreibt, auch immer die Person, die ihn dann publiziert?
- Wie sollen die Empfänger von News gefiltert werden können?
- Gibt es verschiedene Prioritätsstufen für News?

B.1.1.6 Benutzerverwaltung

- Wer ist für die Benutzerverwaltung zuständig?
- Ist es möglich, dass eine Person in mehreren Abteilungen tätig ist?
- Ist es möglich, dass eine Person mehrere Vorgesetzte hat?
- Welche Daten sollen für ein Benutzerprofil erfasst werden (Name, Vorname, E-Mail, etc.)?

B.1.2 Wireframes

B.1.2.1 App

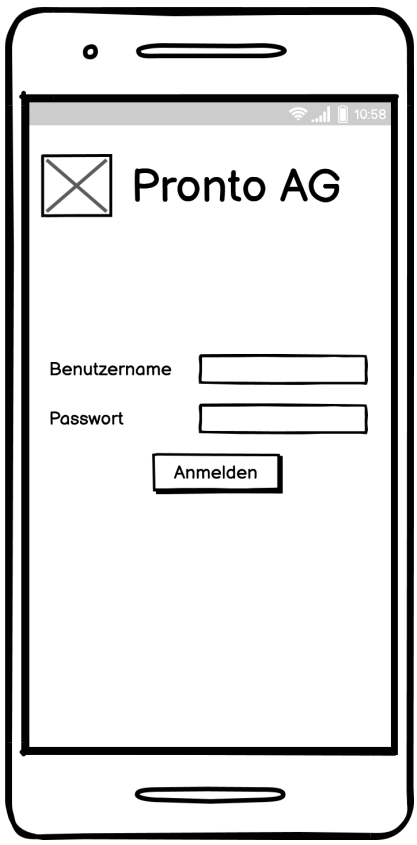

Abbildung B.1: Wireframe Login

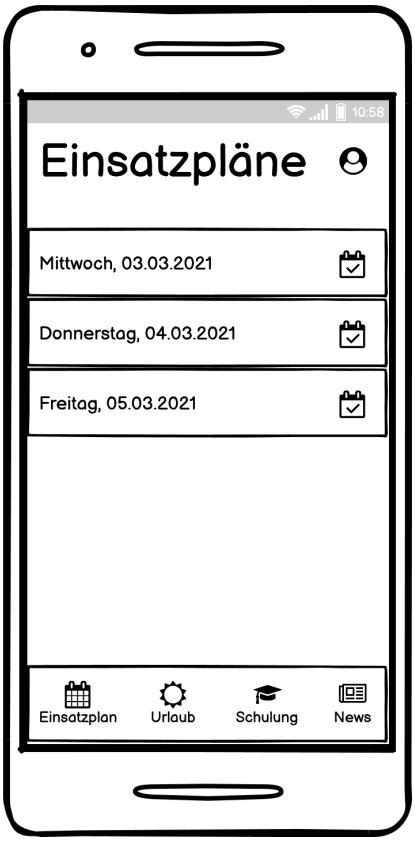

Abbildung B.2: Wireframe Einsatzplanübersicht

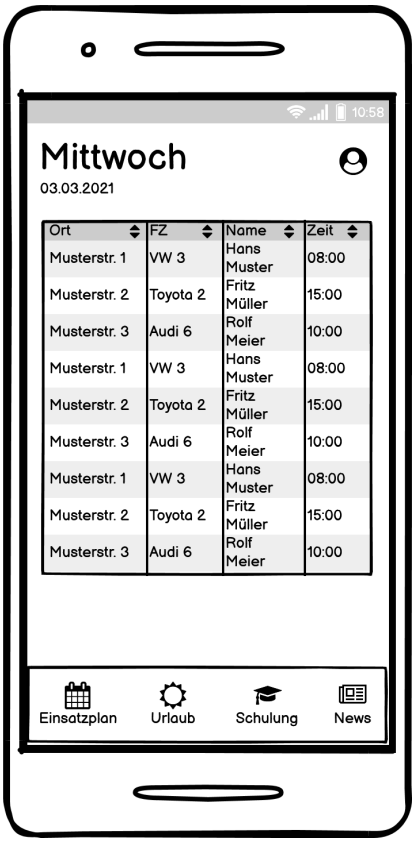

Abbildung B.3: Wireframe Einsatzplandetails

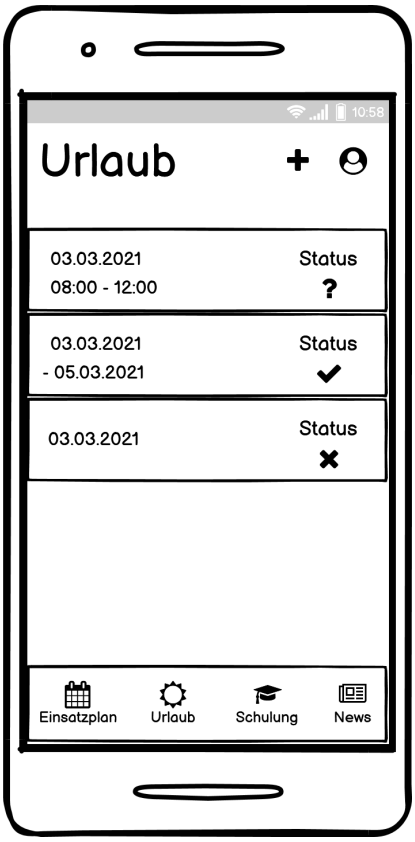

Abbildung B.4: Wireframe Urlaubsübersicht

Abbildung B.5: Wireframe Urlaub Antragstyp

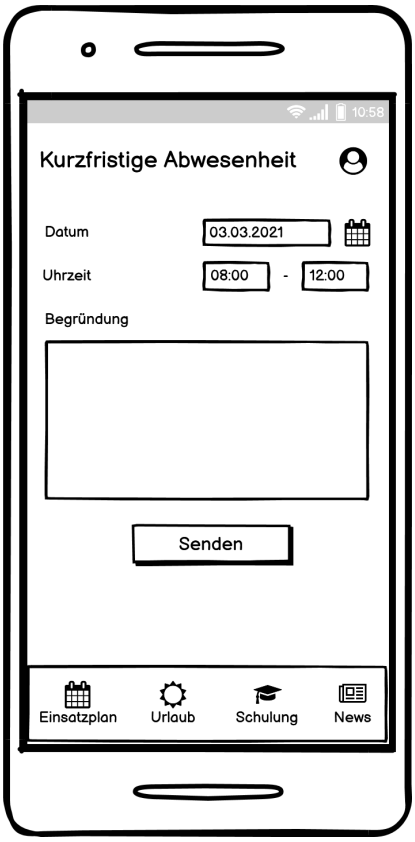

Abbildung B.6: Wireframe Kurzfristige Abwesenheit

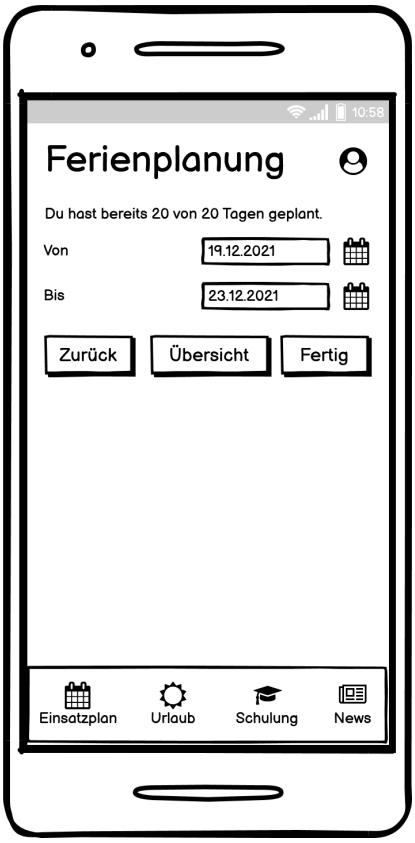

Abbildung B.7: Wireframe Jährliche Urlaubsplanung

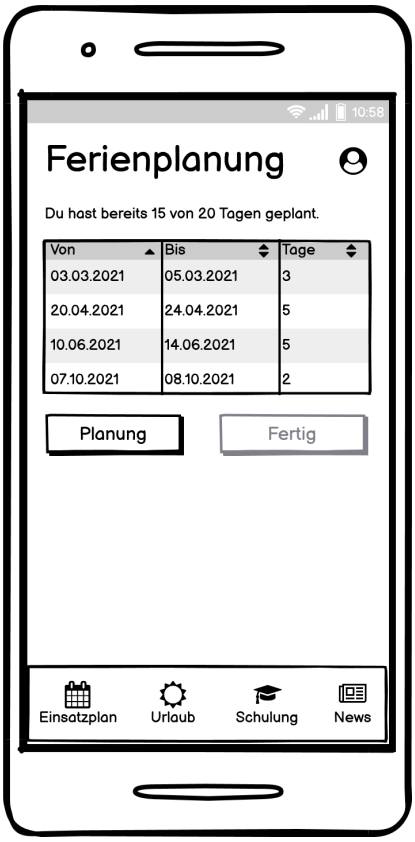

Abbildung B.8: Wireframe Jährliche Urlaubsübersicht

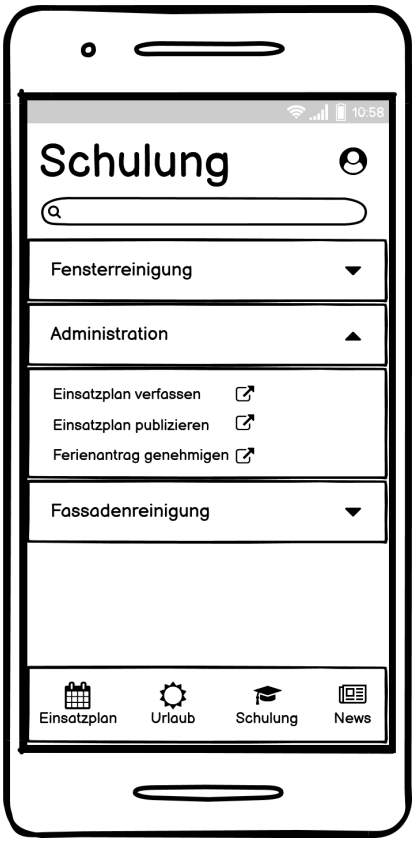

Abbildung B.9: Wireframe Schulungsübersicht

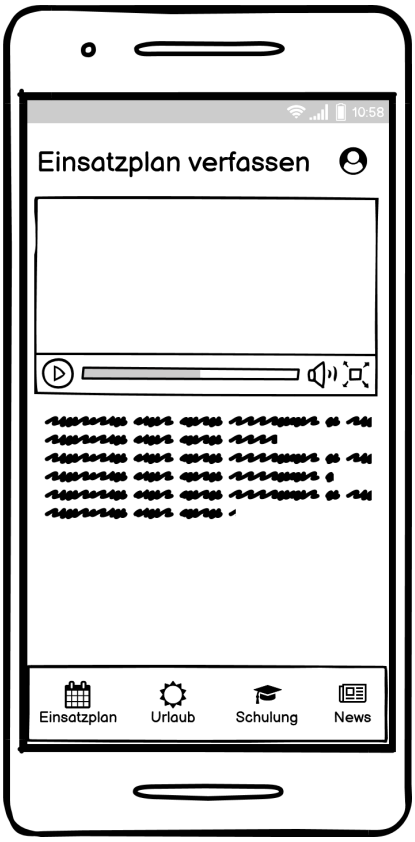

Abbildung B.10: Wireframe Schulungsdetails

Abbildung B.11: Wireframe Newsübersicht

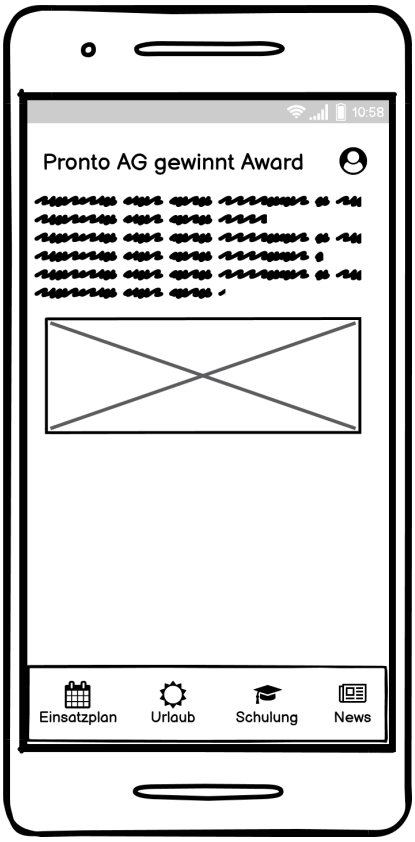

Abbildung B.12: Wireframe Newsdetails

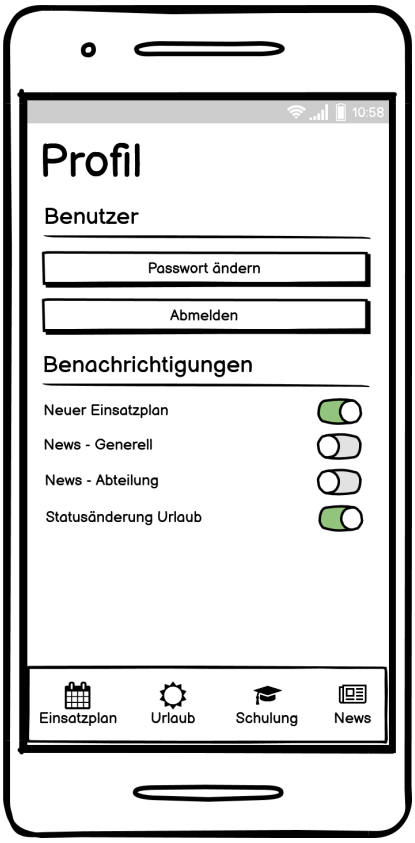

Abbildung B.13: Wireframe Profilübersicht

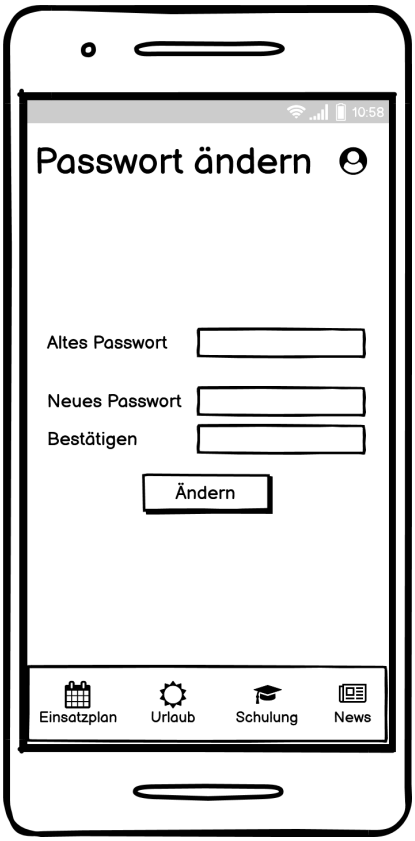

Abbildung B.14: Wireframe Passwort ändern

B.1.2.2 Administrations-Webseite

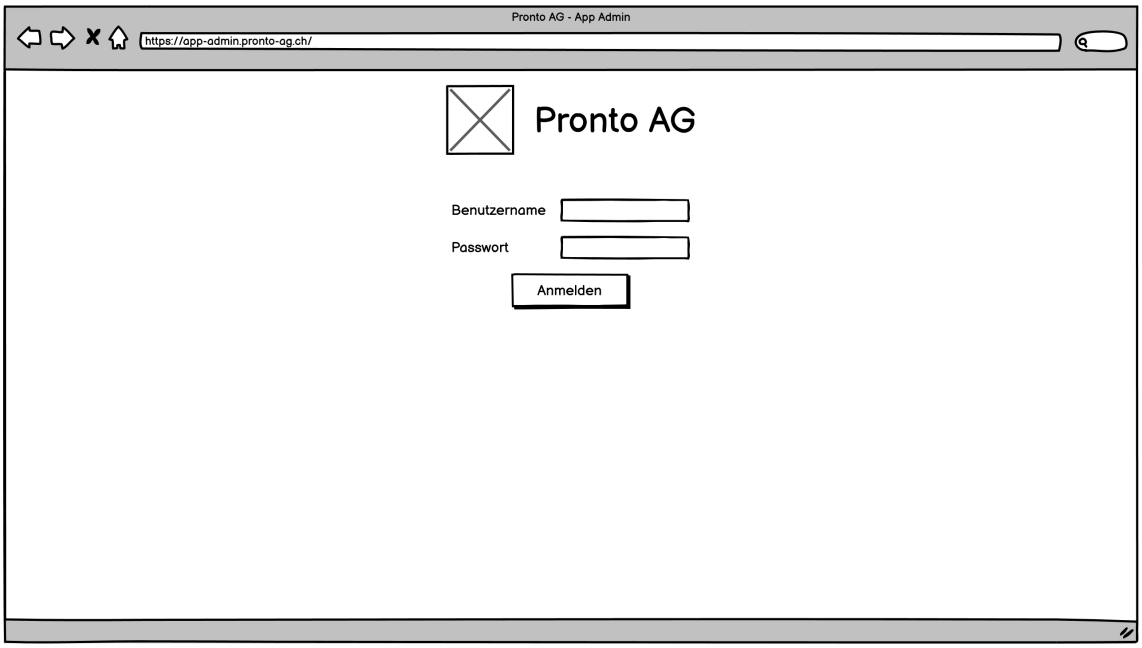

Abbildung B.15: Wireframe Login

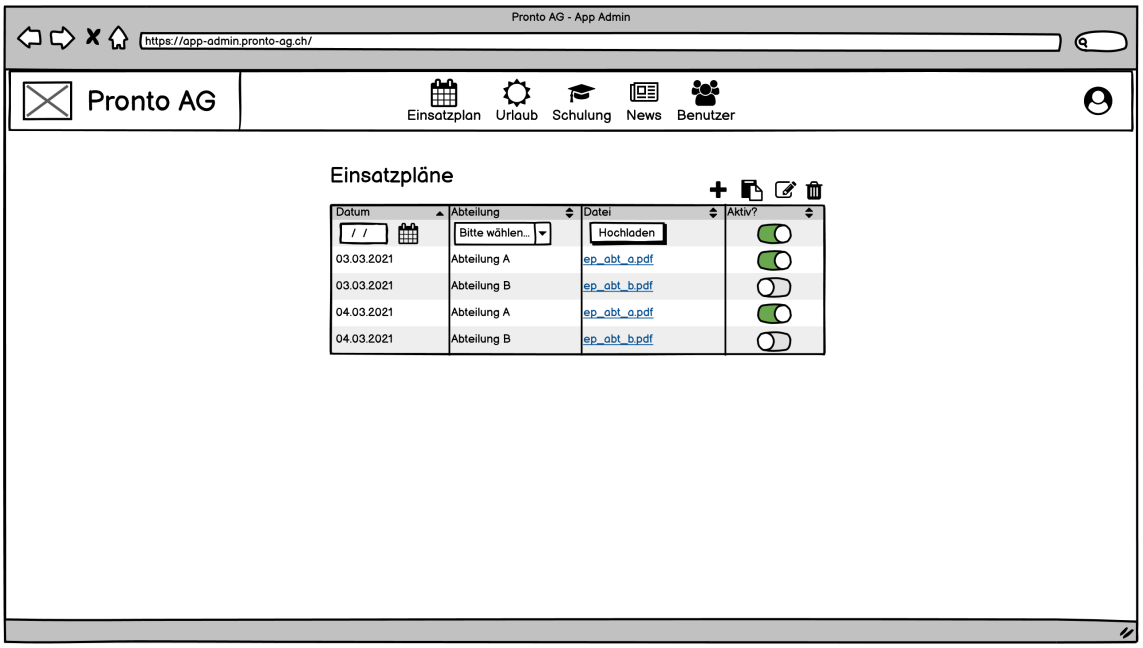

Abbildung B.16: Wireframe Einsatzpläne Ausbaustufe Dateien

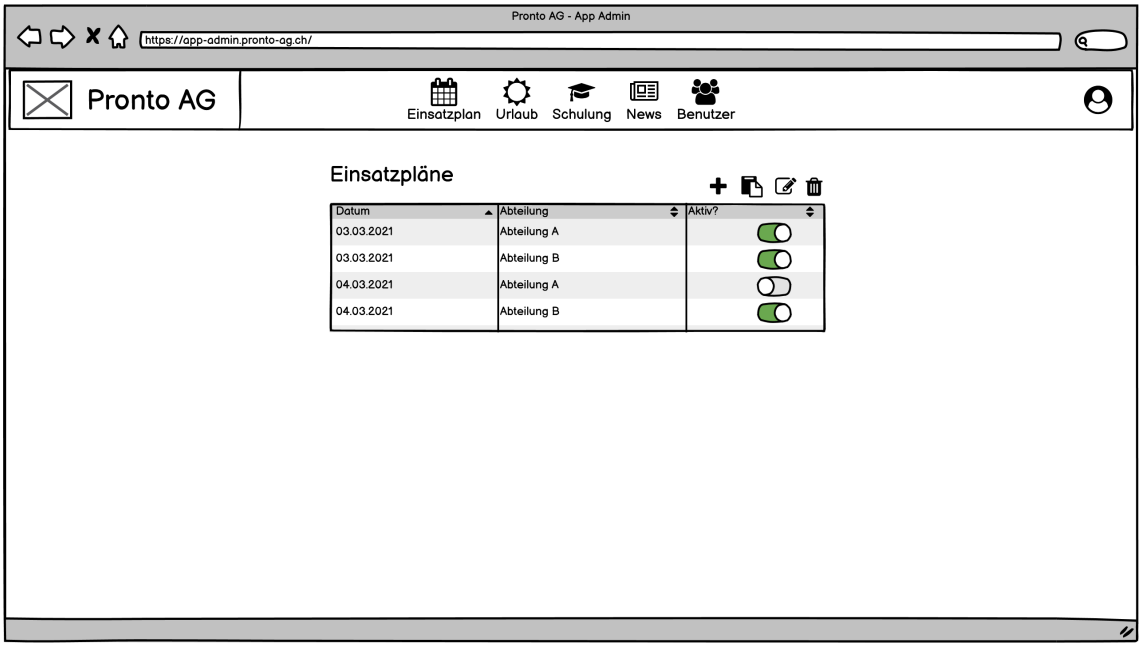

Abbildung B.17: Wireframe Einsatzpläne Ausbaustufe Integriert

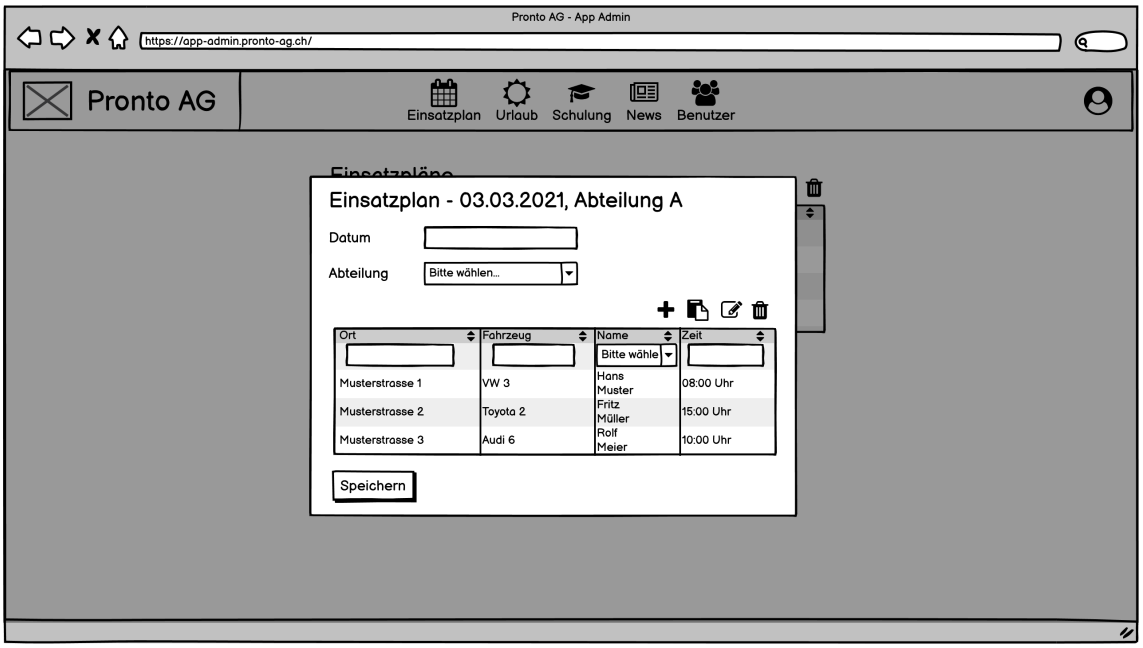

Abbildung B.18: Wireframe Einsatzplan erstellen

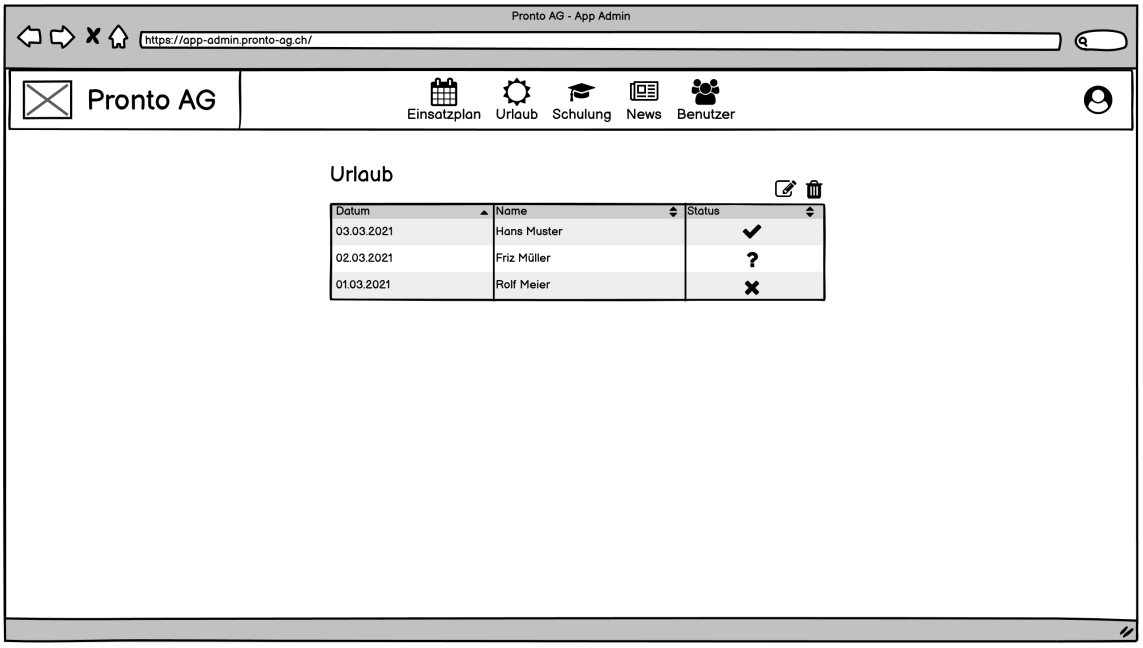

Abbildung B.19: Wireframe Urlaubsverwaltung

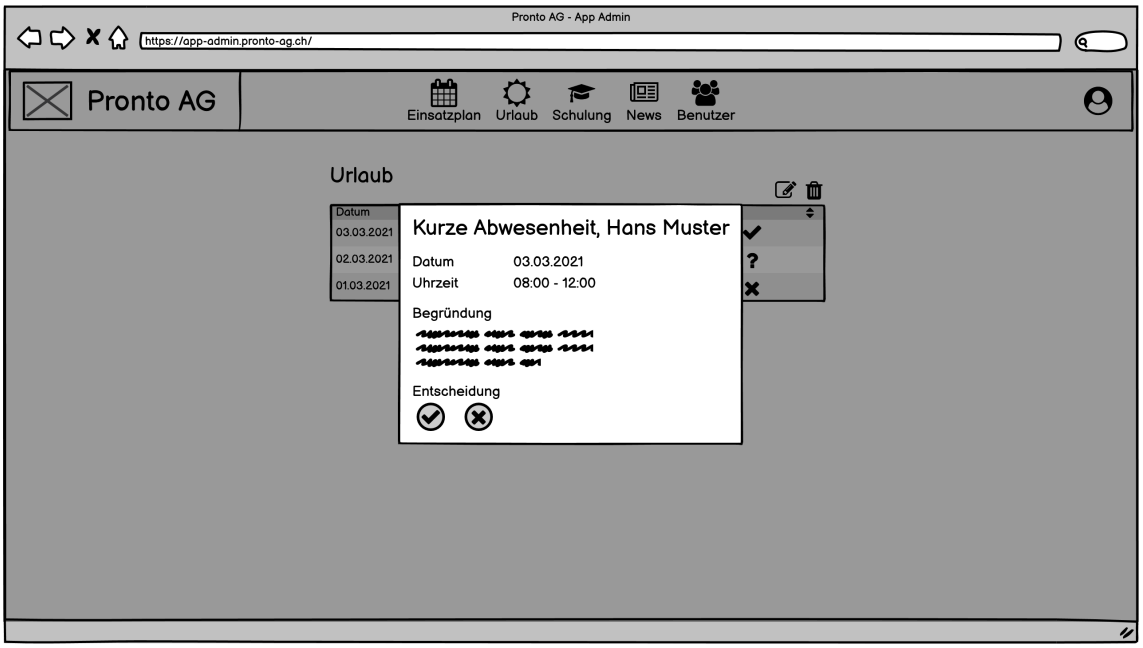

Abbildung B.20: Wireframe Urlaubsantrag

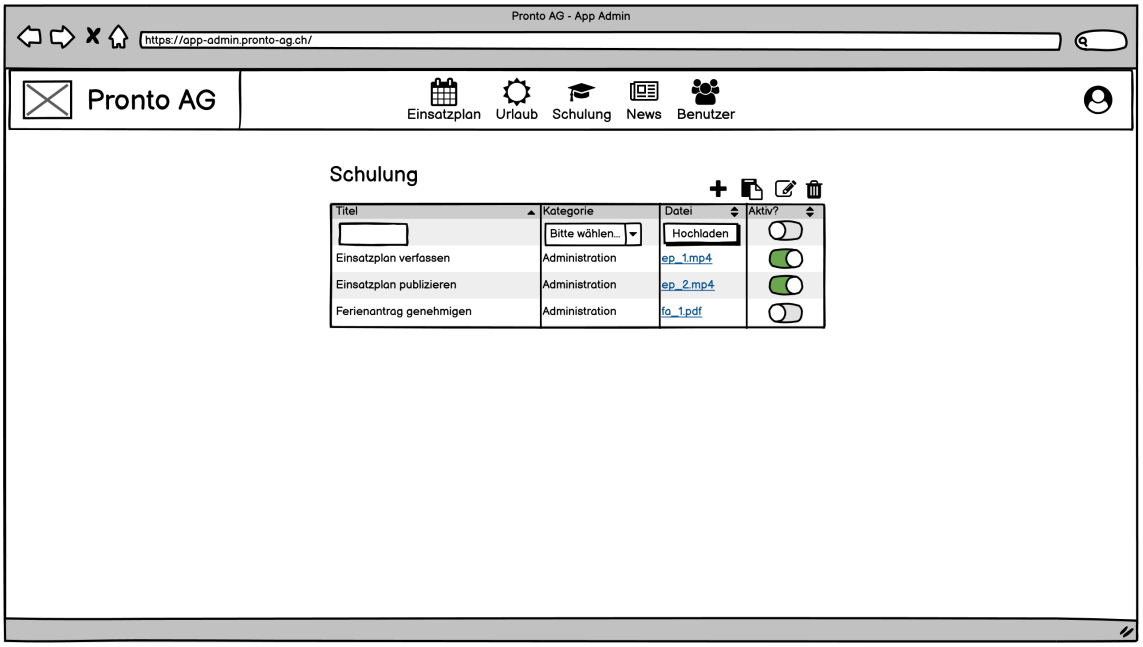

Abbildung B.21: Wireframe Schulungsverwaltung Ausbaustufe Dateien

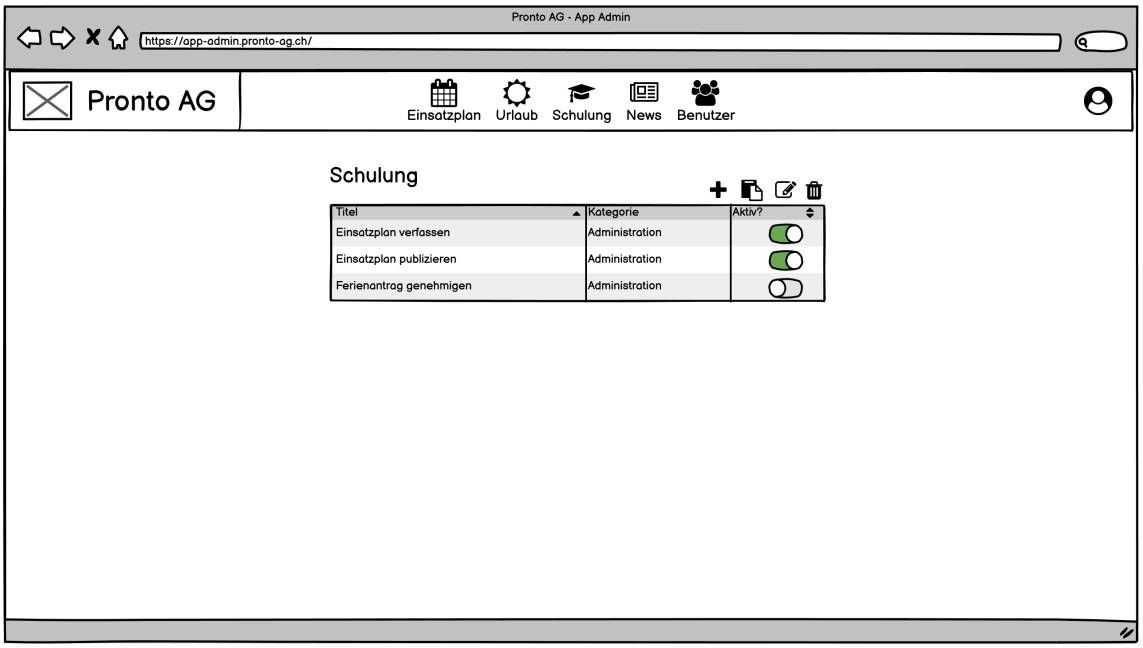

Abbildung B.22: Wireframe Schulungsverwaltung Ausbaustufe Integriert

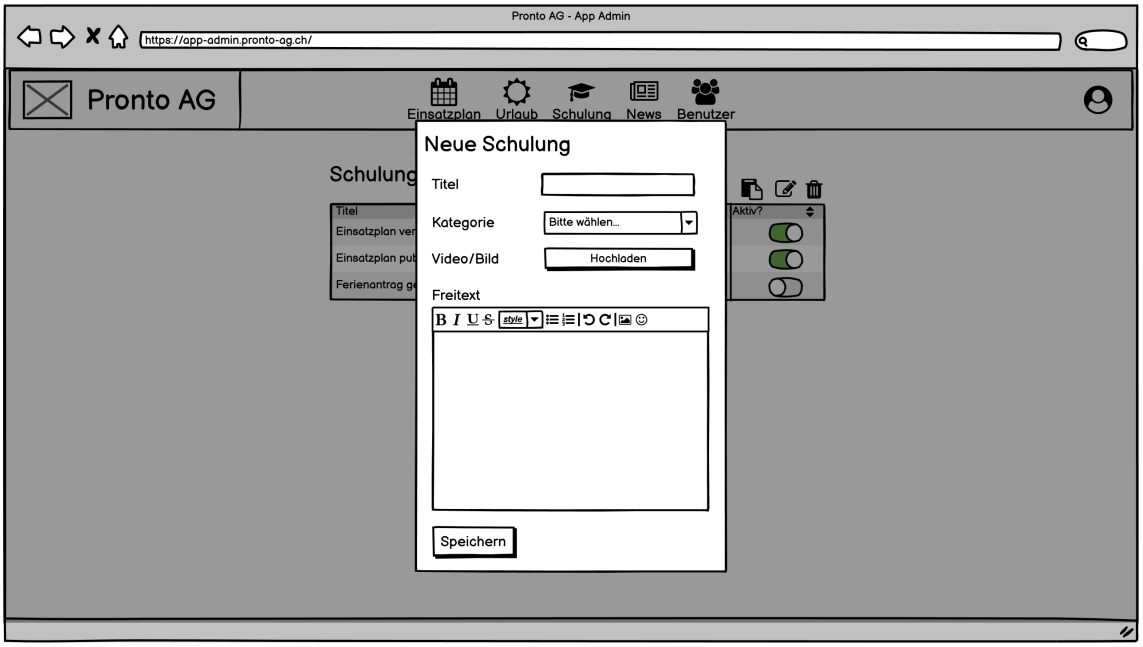

Abbildung B.23: Wireframe Schulungsinhalt erstellen

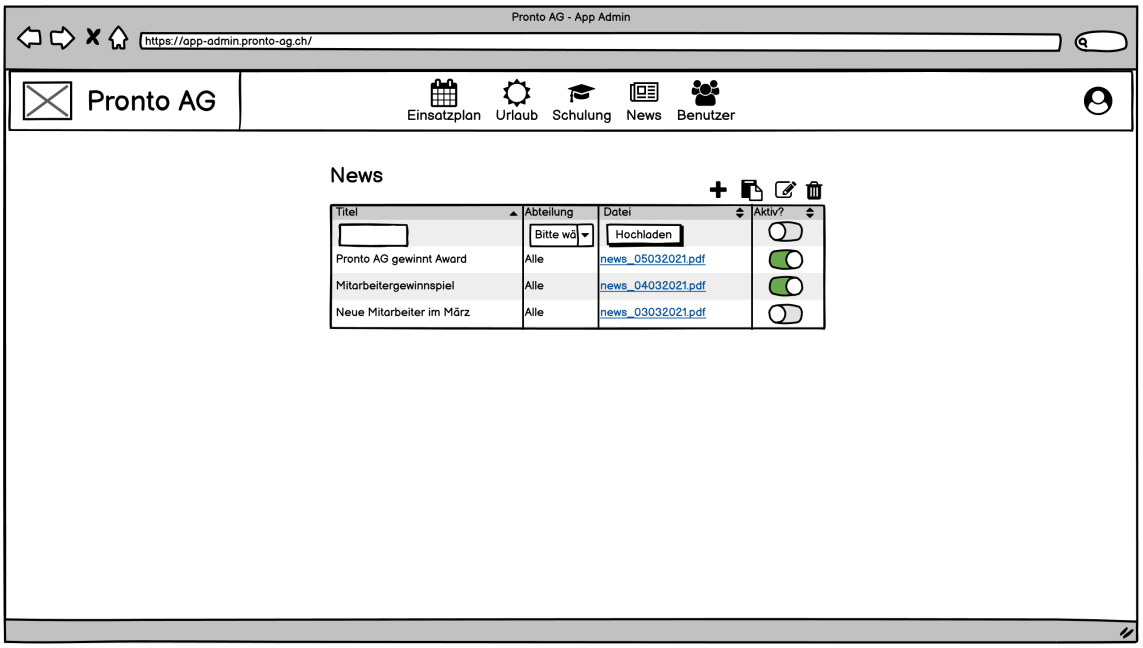

Abbildung B.24: Wireframe Newsverwaltung Ausbaustufe Dateien

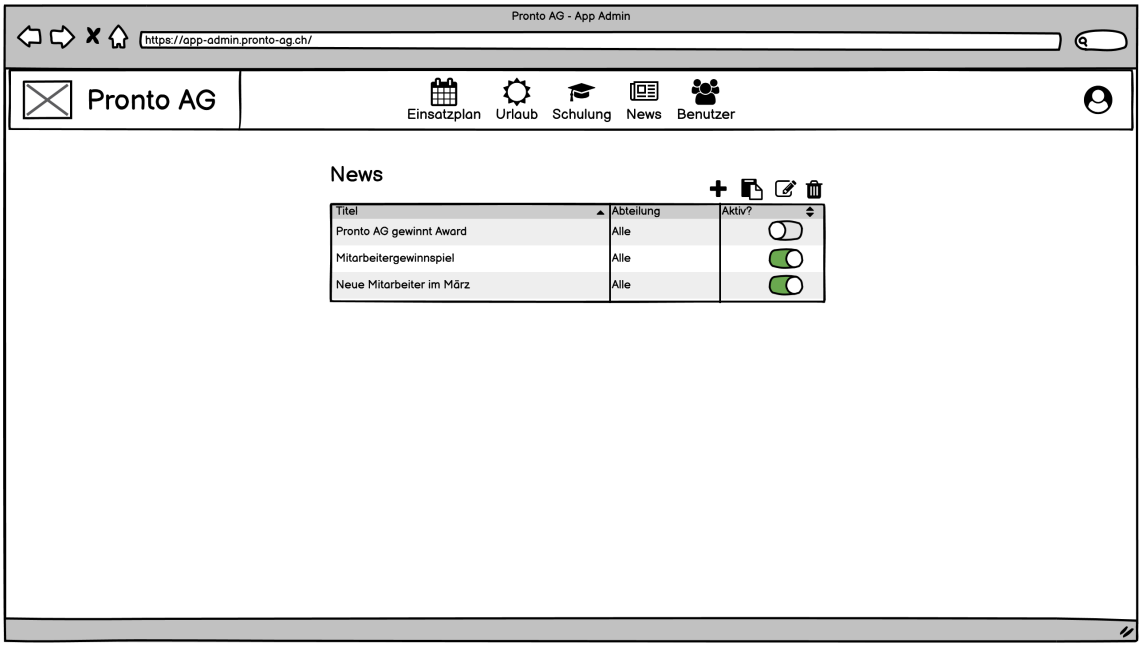

Abbildung B.25: Wireframe Newsverwaltung Ausbaustufe Integriert

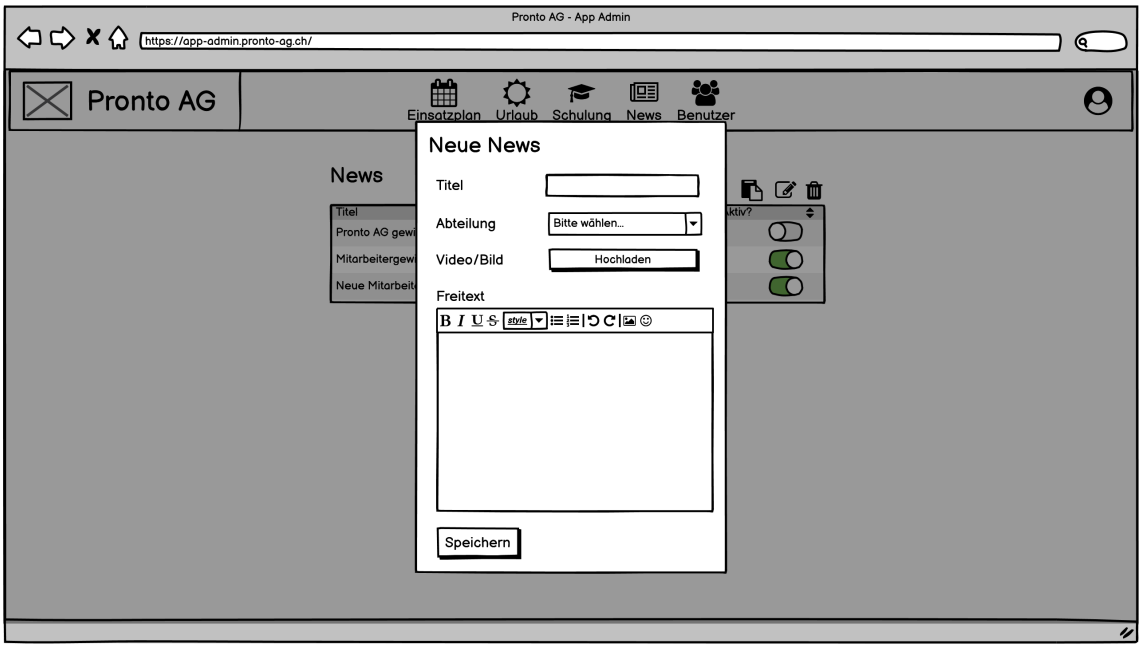

Abbildung B.26: Wireframe Newsartikel erstellen

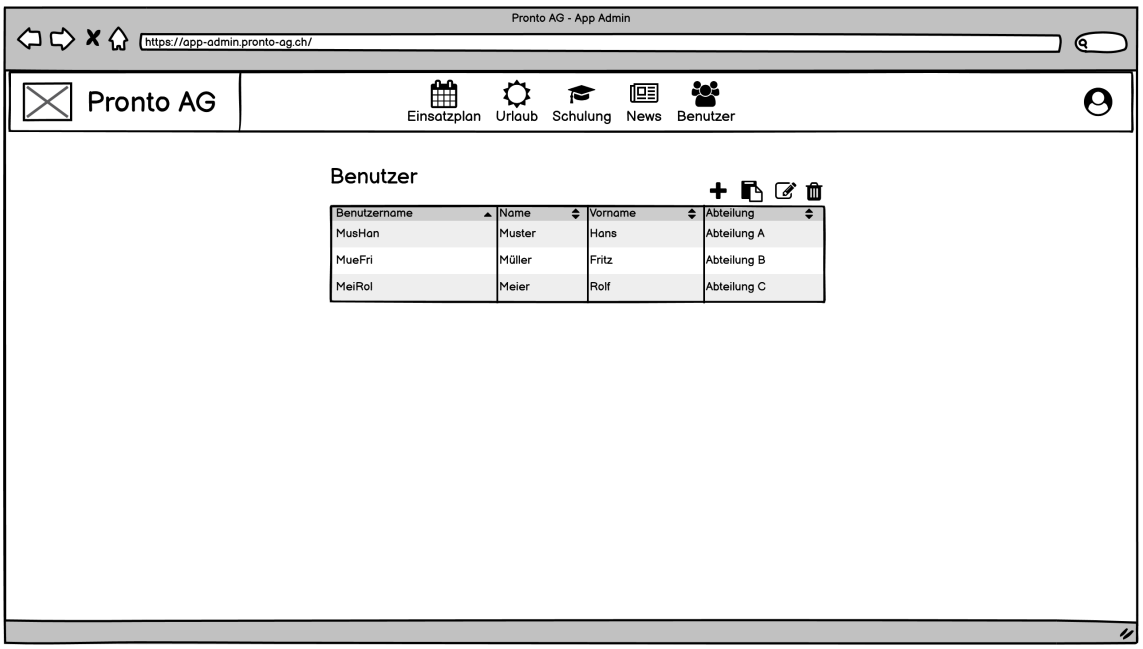

Abbildung B.27: Wireframe Benutzerverwaltung

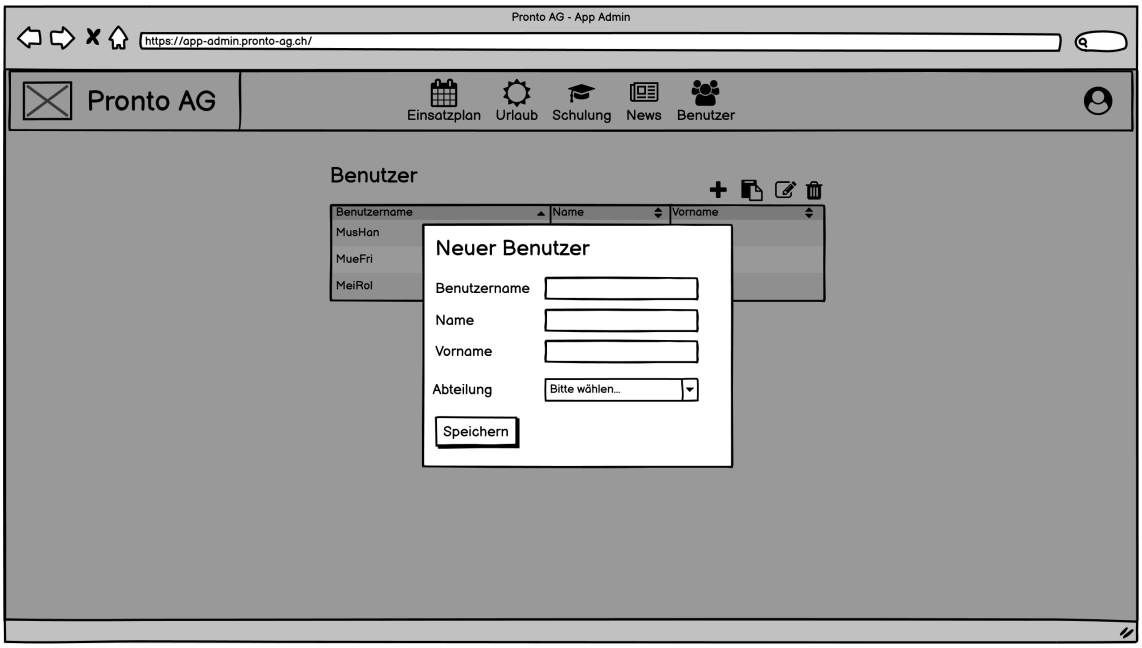

Abbildung B.28: Wireframe Benutzer erstellen

C | Literaturverzeichnis

[1] „Pronto AG Homepage.“ Deutsch, Pronto AG. (2021), Adresse: https://www.pronto-ag.ch/ (besucht am 08. 06. 2021).

[2] „Git Feature Branch Workflow.“ Englisch, Atlassian. (), Adresse: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow (besucht am 15. 06. 2021).

[3] „ISO/IEC 9126-1:2001.“ Englisch, International Organization for Standardization. (2001), Adresse: https://www.iso.org/standard/22749.html.

[4] „Flutter Homepage.“ Englisch, Google. (), Adresse: https://flutter.dev/ (besucht am 03. 03. 2021).

[5] „Flutter push integration.“ Englisch, Google. (7. Okt. 2020), Adresse: https://pub.dev/packages/firebase_messaging (besucht am 03. 03. 2021).

[6] „Flutter FAQ.“ Englisch, Google. (), Adresse: https://flutter.dev/docs/resources/faq (besucht am 03. 03. 2021).

[7] „Flutter FAQ.“ Englisch, Google. (3. März 2021), Adresse: https://developers.googleblog.com/2021/03/announcing-flutter-2.html (besucht am 04. 03. 2021).

[8] „React Native Homepage.“ Englisch, Facebook. (), Adresse: https://reactnative.dev/ (besucht am 03. 03. 2021).

[9] „React Native push integration.“ Englisch, Facebook. (), Adresse: https://reactnative.dev/docs/pushnotificationios (besucht am 03. 03. 2021).

[10] „Xamarin Homepage.“ Englisch, Microsoft. (), Adresse: https://dotnet.microsoft.com/apps/xamarin (besucht am 04. 03. 2021).

[11] „Xamarin 5.“ Englisch, Code Magazine. (27. Juli 2020), Adresse: https://docs.microsoft.com/en-us/azure/developer/mobile-apps/notification-hubs-backend-service-xamarin-forms (besucht am 04. 03. 2021).

[12] „Xamarin 5.“ Englisch, Code Magazine. (21. Dez. 2020), Adresse: https://www.codemag.com/Article/2010082/Xamarin.Forms-5-Dual-Screens-Dark-Modes-Designing-with-Shapes-and-More (besucht am 04. 03. 2021).

[13] „.NET Multi-platform App UI (MAUI).“ Englisch, Microsoft. (4. März 2020), Adresse: https://github.com/dotnet/maui/blob/main/README.md (besucht am 04. 03. 2021).

[14] „Ionic Homepage.“ Englisch, Ionic. (), Adresse: https://ionic.io (besucht am 04. 03. 2021).

[15] „Ionic push integration.“ Englisch, Ionic. (), Adresse: https://ionicframework.com/docs/native/push (besucht am 04. 03. 2021).

[16] „Ionic components.“ Englisch, Ionic. (9. Dez. 2019), Adresse: https://ionicframework.com/docs/components (besucht am 04. 03. 2021).

[17] O. Zimmermann, *INTEGRATION, PART 3: (MICRO-)SERVICES, REST, CONTRACT LANGUAGES, Application Architecture Lesson 11, Lecture Fall Term (dt. Herbstsemester) 2020/2021*, englisch. Rapperswil: Institute for Software, 24. Nov. 2020.

[18] „GraphQL Homepage.“ Englisch, The GraphQL Foundation. (), Adresse: https://graphql.org/ (besucht am 10. 03. 2021).

[19] „GraphQL is the better REST.“ Englisch, Prisma. (23. Feb. 2021), Adresse: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/ (besucht am 10. 03. 2021).

[20] C. Smith. „Advantages and Disadvantages of GraphQL.“ Englisch, Stable Kernel. (), Adresse: https://stablekernel.com/article/advantages-and-disadvantages-of-graphql/ (besucht am 10. 03. 2021).

[21] „gRPC Homepage.“ Englisch, The gRPC Authors & The Linux Foundation. (2021), Adresse: https://grpc.io/ (besucht am 10. 03. 2021).

[22] J. Newton-King. „Compare gRPC services with HTTP APIs.“ Englisch, Microsoft. (12. Mai 2019), Adresse: https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-5.0 (besucht am 10. 03. 2021).

[23] „Simple app state management.“ Englisch, Flutter. (), Adresse: https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple (besucht am 07. 06. 2021).

[24] „Navigator class.“ Englisch, Flutter. (27. Mai 2021), Adresse: https://api.flutter.dev/flutter/widgets/Navigator-class.html (besucht am 07. 06. 2021).

[25] J. Seric. „GraphQL multipart request specification.“ Englisch. (7. Juni 2021), Adresse: https://github.com/jaydenseric/graphql-multipart-request-spec (besucht am 08. 06. 2021).

[26] „Hot Chocolate Github.“ Englisch, ChilliCream. (8. Juni 2021), Adresse: https://github.com/ChilliCream/hotchocolate (besucht am 08. 06. 2021).

[27] „FirebaseAdmin.Messaging.FirebaseMessaging.“ Englisch, Google. (9. Sep. 2020), Adresse: https://firebase.google.com/docs/reference/admin/dotnet/class/firebase-admin/messaging/firebase-messaging#class_firebase_admin_1_1_messaging_1_1_firebase_messaging_1ac4b7d71e8a5ece162524cba1d2bef82e (besucht am 08. 06. 2021).

[28] J. Ryan. „Learning Flutters new navigation and routing system.“ Englisch. (30. Sep. 2020), Adresse: https://medium.com/flutter/learning-flutters-new-navigation-and-routing-system-7c9068155ade (besucht am 10. 06. 2021).

[29] „Stacked.“ Englisch, FilledStacks. (9. Juni 2021), Adresse: https://pub.dev/packages/stacked (besucht am 11. 06. 2021).

[30] D. Roccasalva. „Stealing JWTs in localStorage via XSS.“ Englisch. (10. Sep. 2020), Adresse: https://medium.com/redteam/stealing-jwts-in-localstorage-via-xss-6048d91378a0 (besucht am 11. 06. 2021).

[31] C. Zaccagnino. „Securely Storing JWTs in (Flutter) Web Apps.“ Englisch. (29. Apr. 2020), Adresse: https://carmine.dev/posts/flutterwebjwt/ (besucht am 10. 06. 2021).

[32] P. Kellner. „Log Your Queries While Building a GraphQL Server.“ Englisch. (10. Jan. 2021), Adresse: https://chillicream.com/blog/2021/01/10/hot-chocolate-logging (besucht am 10. 06. 2021).

[33] „Log Scopes.“ Englisch, Microsoft. (28. Nov. 2020), Adresse: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-5.0#log-scopes (besucht am 15. 06. 2021).
[34] „Admin SDK Reference.“ Englisch, Google. (1. Juni 2021), Adresse: https://firebase.google.com/docs/reference/admin (besucht am 10. 06. 2021).
[35] „Mockito - Null Safety.“ Englisch, Dart. (2. Juni 2021), Adresse: https://github.com/dart-lang/mockito/blob/master/NULL_SAFETY_README.md#fallback-generators (besucht am 15. 06. 2021).

D Abbildungsverzeichnis

E Tabellenverzeichnis

F | Auflistungsverzeichnis

G | Glossar

LaTeX

Textsystem zur Erstellung von technischen und wissenschaftlichen Dokumenten. Verfügbar unter https://www.latex-project.org/ 23

API

API steht für "application programming interface „ was auf Deutsch übersetzt Programmierschnittstelle bedeutet. Die API fungiert als Anbindungsstelle, über welche andere Software auf die angebotenen Funktionen zugreifen kann. Mehr dazu: https://de.wikipedia.org/wiki/Programmierschnittstelle 9, 42, 56–58, 63–65, 67

Beta-Phase

Wenn sich eine Applikation in der Beta-Phase befindet so bedeutet das, dass diese noch im Entwicklungsstadium ist. Im Gegensatz zur Alpha-Phase sind in der Beta-Phase eigentlich alle Funktionalitäten vorhanden allerdings nicht vollständig getestet. Es können entsprechend noch Fehler bei der Benutzung auftreten. Die Beta-Phase wird oft als ein letzter Test durch Benutzer vor der eigentlichen Veröffentlichung genutzt. Mehr dazu: https://de.wikipedia.org/wiki/Entwicklungsstadium_(Software)#Beta-Version 51

Clockify

Tool für die Zeitschätzung und Erfassung. Verfügbar unter https://clockify.me. 22–24

cross-platform

Cross-platform ist ein Begriff aus der Informatik, welcher Software beschreibt, die auf mehreren Plattformen (Geräten, Betriebssystemen) lauffähig ist. 1, 12, 13, 21, 44, 50–53, 64

Definition of Done

Eine Liste von Aufgaben, welche erledigt sein müssen, bevor ein Projekt oder Arbeitspaket als abgeschlossen angesehen werden kann. 22, 25

End of Elaboration

Meilenstein in der RUP-Methodik, bei welchem die grössten Risiken behoben sein müssen. Bei diesem Meilenstein wird entschieden, ob mit dem Projekt so fortgefahren werden kann oder nicht. 15, 21

End of Elaboration Checklist

Eine Liste von Bedingungen, welche zum End of Elaboration erfüllt worden sein müssen. Die Bedingungen sind wie folgt:

- Wir haben den Kunden verstanden (Requirements so vollständig wie nur möglich). Scope (=grober Funktionsumfang) ist vereinbart.
- Wir haben alle Werkzeuge im Griff (IDE, Versionskontrolle, Build Server, Deployment, Unit Testing, Workflow Tools, Wiki, ...)
- Wir wissen, wie die Architektur aussehen wird (Architektur skizziert, die grossen Interfaces festgelegt, Architektur-Prototypen gemacht).
- Für das User Interface gibt es einen ersten Entwurf (Grafiken,Wireframes, klickbarer Prototyp) und dem Kunden gefällt es.
- Wir haben die Marschrichtung (grobe Meilensteine) für die nächsten zwei Iterationen festgelegt, viele Arbeitspakete erstellt, und dem Kunden eine genauere Zeitschätzung geliefert.
- Alle grossen Risiken, alle grossen Fragezeichen sind weg.

15, 17

Feature Branch Workflow

Workflow in Git, bei welchem die Entwicklung von Features in separaten Branches erfolgt. Dokumentiert unter https://www.atlassian.com/git/tutorials/comparing-workflo feature-branch-workflow 21

Feature-Freeze

Zeitpunkt in einem Projekt, ab welchem keine neuen Features mehr implementiert werden. 15, 16

Framework

Ein Framework ist ein Programmiergerüst und stellt einen Rahmen zur Verfügung, in welchem ein Entwickler seine Anwendung programmieren kann. Das Framework gibt hierbei die Rahmenbedinungen vor und stellt Funktionalitäten zur Verfügung. Mehr dazu: https://de.wikipedia.org/wiki/Framework 1, 28, 29, 44, 45, 50–56, 64–67

FURPS

FURPS ist ein Modell aus dem Bereich der Softwarequalität. Es entstammt dem Englischen und fasst Qualitätsmerkmale von Software zusammen. Siehe auch https://en.wikipedia.org/wiki/FURPS 41

GitHub

Dienst zur Versionsverwaltung von Softwareprojekten. Verfügbar unter https://github.com/. 23–25

HATEOAS

HATEOAS steht für „Hypermedia as the Engine of Application State“. Diese Einschränkung von REST besagt, dass ein REST-Client wenig bis gar keine Ahnung von der Applikation haben muss, wenn er darauf zugreift. Die Applikation soll dem Client nach jeder Anfrage links zur Verfügung stellen, mit welchen dieser alle anderen möglichen nachfolgenden Operationen durchführen kann. Dadurch kann der Client sich mithilfe der Links durch die Applikation bewegen. Mehr dazu: https://de.wikipedia.org/wiki/HATEOAS 57

Linter

Software zur statischen Code-Analyse. Wird oft zur Überprüfung der Code-Formatierung verwendet. Mehr dazu: https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug) 25

McCabe-Wert

Der McCabe-Wert auch „Cyclomatic Complexity“ genannt ist eine Messgrösse in der Softwareentwicklung, mithilfe welcher die Komplexität einer Software angegeben werden kann. Siehe auch: https://de.wikipedia.org/wiki/McCabe-Metrik 42

Microsoft Teams

Platform für Kommunikation in Form von Chats und Anrufen. Verfügbar unter https://www.microsoft.com/de-ch/microsoft-teams/group-chat-software. 23

MIT-Lizenz

Die MIT-Lizenz ist eine Open-Source-Softwarelizenz. Sie ermöglicht die Wiederverwendung der Software in Open-Source und Closed-Source Projekten. Der Lizenztext ist verfügbar unter: https://opensource.org/licenses/MIT 33

NFR

Abkürzung für den englischen Begriff ”non-functional requirement”, was auf Deutsch nichtfunktionale Anforderung heisst. 41, 47

Overfetching

Overfetching tritt auf, wenn ein Client Daten abfragt, welche er eigentlich nicht braucht. Dies ist häufig bei REST der Fall, da ein Endpunkt eine fixe Daten zurückliefert, welche eventuell nicht komplett gebraucht werden. 58, 59

potentially shippable product

Softwareprodukt, welches in dieser Form an den Kunden ausgeliefert werden könnte. Alle implementierten Features sind komplett funktionsfähig. 14

Push-Benachrichtigung

Unter einer Push-Benachrichtigung wird in dieser Arbeit eine Nachricht verstanden, welche vom Server an das Endgerät (Mobiltelefon/Browser) geschickt wird. Das Endgerät verarbeitet anschliessend diese Nachricht. Speziell ist hierbei, dass das Endgerät die Nachrichten nicht beim Server abholen muss, sondern diese zugeschickt bekommt. 1, 9, 12, 28, 36, 47, 50–55, 63

RUP

Rational Unified Process 21

Scrum +

Projektmethodik welche Scrum mit dem End of Elaboration von RUP verbindet. 14, 21

Semantic Versioning

Standard zur Versionierung von Software, dokumentiert unter https://semver.org/lang/de/ 47

Service Locator

Service Locator beschreibt ein Architektur-Pattern in der Softwareentwicklung, bei welchem die Aufgabe Services zu verwalten an ein zentrales Register übergeben wird. Clients der Services haben die Möglichkeit, über den Service Locator, auf die Services zuzugreifen. Mehr dazu: https://www.geeksforgeeks.org/service-locator-patter
65

Single-Responsibility-Prinzip

Ein Prinzip aus der Softwareentwicklung welches besagt, dass Softwarekomponenten nur eine fest definierte Aufgabe zu erfüllen haben. Mehr dazu: https://de.wikipedia.org/wiki/Single-Responsibility-Prinzip 61

Underfetching

Underfetching tritt auf, wenn ein Client in einer Anfrage weniger Daten erhält, als dieser eigentlich benötigt. Dies ist häufig bei REST der Fall, da ein Endpunkt lediglich für eine Ressource genutzt wird. Aufgrund dessen sind häufig mehrere Abragen notwendig um Daten von abhängigen Ressourcen zu erhalten. 58, 59

Usability-Test

Ein Usability-Test ist ein Test, welcher mit Benutzern einer Applikation durchgeführt wird, um deren Bedienbarkeit für den Endbenutzer zu evaluieren. Mehr dazu: https://de.wikipedia.org/wiki/Usability-Test 15, 26

WYSIWYG

WYSIWYG steht für "What you see is what you get" und wird häufig auch Echtzeitdarstellung genannt. Der Begriff wird häufig bei Programmen verwendet, welche dem Benutzer während der Bearbeitung bereits anzeigen, wie das Endprodukt aussieht. 38, 39

Xamarin.Forms

Ist eine UI-Bibliothek welche auf Xamarin aufsetzt und es ermöglicht für die verschiedenen Betriebssysteme die selben UI-Komponenten zu verwenden. 53, 54

H Schema GraphQL-API

```
schema {
  query: Query
  mutation: Mutation
}

"Class containing the access control list of a specific user."
type AccessControlList {
  "Gets or sets the acl identifier."
  id: Int!
  "Gets or sets the user associated with this list."
  user: User
  "Gets or sets a value indicating whether the user has permission to\nview all users."
  canViewUsers: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nview all users of his department."
  canViewDepartmentUsers: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nedit all users."
  canEditUsers: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nedit all users of his department."
  canEditDepartmentUsers: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nview all departments."
  canViewDepartments: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nview his own department."
  canViewOwnDepartment: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nedit all departments."
  canEditDepartments: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nedit his own department."
  canEditOwnDepartment: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nview all deployment plans."
  canViewDeploymentPlans: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nview all deployment plans of his department."
  canViewDepartmentDeploymentPlans: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nedit all deployment plans."
  canEditDeploymentPlans: Boolean!
  "Gets or sets a value indicating whether the user has permission to\nedit all deployment plans of his department."
  canEditDepartmentDeploymentPlans: Boolean!
}

"Class representing a department."
type Department {
  "Gets or sets the id of the department."
  id: Int!
  "Gets or sets the name of the department."
  name: String
  "Gets or sets list of User belonging to this\ndepartment."
  users: [User]
  "Gets or sets list of DeploymentPlan belonging to this\ndepartment."
  deploymentPlans: [DeploymentPlan]
  "Gets the id of this department. Needed for\nIDepartmentComparable interface."
  departmentId: Int
}

"Class that contains graphQL extensions for the deployment plan object."
```

```
type DeploymentPlan {
  "Gets or sets the id used as primary key by the database."
  id: Int!
  "Gets or sets the description which may be used to identify the\ndeployment plan."
  description: String
  "Gets or sets the moment from which this deployment plan should be\ntreated as active."
  availableFrom: DateTime!
  "Gets or sets the moment until which the deployment plan will be\ntreated as active."
  availableUntil: DateTime!
  "Gets or sets a value indicating whether the deployment plan is in\npublished status. Unpublished deployment plans should not be shown\nto users without administrative permissions."
  published: Boolean!
  "Gets or sets the department associated with this deployment\nplan."
  department: Department @authorize(policy: "ViewDepartment", apply: BEFORE_RESOLVER)
  "Gets the link to the file associated with this deployment plan.\n\n\n**Returns:**\nThe link to where the file associated with this deployment\nplan is statically served."
  link: Url
}

"Token representing a device registered with firebase cloud messaging."
type FcmToken {
  "Gets or sets the token identifier."
  id: String
  "Gets or sets the user associated with this token."
  owner: User
}

"Class representing the mutation operation of graphql.\nContains all mutations that concern Department\nobjects."
type Mutation {
  "Adds a deployment plan to the application.\n\n\n**Returns:**\nThe newly generated deployment plan."
  createDeploymentPlan("The file to be associated with the new deployment\n plan." file: Upload! "The moment from which the deployment\n plan will be treated as active." availableFrom: DateTime! "The moment until which the deployment\n plan will be treated as active." availableUntil: DateTime! "The id of the department the new\n deployment plan should be linked to." departmentId: Int! "Short description to identify the\n deployment plan." description: String): DeploymentPlan @authorize(policy: "EditDeploymentPlan", apply: BEFORE_RESOLVER)
  "Method that updates the deployment plan with the given id according\nto the provided information.\n\n\n**Returns:**\nThe updated deployment plan."
  updateDeploymentPlan("The id of the deployment plan to be adjusted." id: Int! "The file to be associated with the new deployment\n plan." file: Upload "The moment from which the deployment\n plan will be treated as active." availableFrom: DateTime "The moment until which the deployment\n plan will be treated as active." availableUntil: DateTime "The id of the department the\n deployment plan should be linked to." departmentId: Int "Short description to identify the\n deployment plan." description: String): DeploymentPlan! @authorize(policy: "EditDeploymentPlan", apply: BEFORE_RESOLVER)
  "Method that publishes the deployment plan with the given id. If\nthe deployment plan is already published nothing will be done.\n\n\n**Returns:**\nTrue if the deployment plan status was updated false if\nthe deployment plan was already published."
  publishDeploymentPlan("The id of the deployment plan to be published." id: Int! "The title of the info notification to be sent." title: String! "The body of the info notification to be sent." body: String!): Boolean! @authorize(policy: "EditDeploymentPlan", apply: BEFORE_RESOLVER)
  "Method that hides the deployment plan with the given id. If the\ndeployment plan is already not in the published state nothing will\nbe done.\n\n\n**Returns:**\nTrue if the deployment plan status was updated false if\nthe deployment plan was already hidden."
  hideDeploymentPlan("The id of the deployment plan to be hidden." id: Int!): Boolean! @authorize(policy: "EditDeploymentPlan", apply: BEFORE_RESOLVER)
  "Method that removes the deployment plan with the given id.\n\n\n**Returns:**\nThe id of the plan which was removed."
  removeDeploymentPlan("Id of the plan to be removed." id: Int!): Int! @authorize(policy:
```

```
"EditDeploymentPlan", apply: BEFORE_RESOLVER)
  "Registers a fcm token for the authenticated user. If the token\nalready exists it will be overwritten with the currently\nauthenticated user.\n\n\n**Returns:**\nTrue if the token was saved successfully."
  registerFcmToken("The token to be registered." fcmToken: String): FcmToken @authorize(apply: BEFORE_RESOLVER)
  "Unregisters a given fcm token from the user it was assigned to.\nIf the token cannot be found nothing will be done.\n\n\n**Returns:**\nTrue if the token could be removed."
  unregisterFcmToken("The fcm token to be removed." fcmToken: String): Boolean!
  "Change the password of an existing user.\n\n\n**Returns:**\nThe newly valid jwt token for the user."
  changePassword("The current password of the user." oldPassword: String! "The new password of the user." newPassword: String!): String! @authorize(apply: BEFORE_RESOLVER)
  "Creates a new User.\n\n\n**Returns:**\nThe newly created user."
  createUser("The username of the new user." userName: String! "The password of the new user." password: String! "The access control list\n            that should be linked to the new user." accessControlList: AccessControlListInput! "The id of the department the user\n should be added to." departmentId: Int!): User! @authorize(policy: "EditUser", apply: BEFORE_RESOLVER)
  "Method that removes the user with the given id.\n\n\n**Returns:**\nThe id of the user which was removed."
  removeUser("Id of the user to be removed." id: Int!): Int! @authorize(policy: "EditUser", apply: BEFORE_RESOLVER)
  "Method that updates the user with the given id according\nto the provided information.\n\n\n**Returns:**\nThe updated user."
  updateUser("Id of the user to be updated." id: Int! "The new username of the user." userName: String "The new password for the user." password: String "The new\n AccessControlList that will be linked to the user." accessControlList: AccessControlListInput "The id of the new department of the\n            user." departmentId: Int): User! @authorize(policy: "EditUser", apply: BEFORE_RESOLVER)
  "Method to create a department.\n\n\n**Returns:**\nThe newly created department."
  createDepartment("The name of the new department." name: String!): Department! @authorize(policy: "EditDepartment", apply: BEFORE_RESOLVER)
  "Method to update a department.\n\n\n**Returns:**\nThe updated department."
  updateDepartment("The id of the department." id: Int! "The new name of the department." name: String): Department! @authorize(policy: "EditDepartment", apply: BEFORE_RESOLVER)
  "Method to remove a department.\n\n\n**Returns:**\nThe id of the removed department."
  removeDepartment("The id of the department." id: Int!): Int! @authorize(policy: "EditDepartment", apply: BEFORE_RESOLVER)
}

"Class representing the mutation operation of graphql.\nContains all mutations that concern Department\nobjects."
type Query {
  "Method which retrieves the available deployment plans. Depending\non the requesting users access rights only a fraction of the\navailable plans might be returned.\n\n\n**Returns:**\nQueryable of all deployment plans available to the user."
  deploymentPlans(where: DeploymentPlanFilterInput order: [DeploymentPlanSortInput!]): [DeploymentPlan] @authorize(policy: "ViewDeploymentPlan", apply: BEFORE_RESOLVER)
  "Method which allows the user to retrieve a token which may then be\nused for authentication in further requests.\n\n\n**Returns:**\nA JWT-Bearer-Token which can be used within the\nauthentication header in order to authenticate the user."
  authenticate("The name of the user wanting to authenticate." userName: String "The password of the user." password: String): String
  "Method which returns the user making the request.\n\n\n**Returns:**\nThe user requesting this endpoint."
  user: User @authorize(apply: BEFORE_RESOLVER)
  "Method which retrieves the available users. Depending\non the requesting users access rights only a fraction of the\navailable users might be returned.\n\n\n**Returns:**\nQueryable of all users available to the requesting\nuser."
  users(where: UserFilterInput order: [UserSortInput!]): [User] @authorize(policy: "ViewUser", apply: BEFORE_RESOLVER)
  "Method which retrieves the available departments. Depending\non the requesting users access rights only a fraction of the\navailable departments might be returned.\n\n\n**Returns:**\nQueryable of all departments available to the user."
  departments(where: DepartmentFilterInput order: [DepartmentSortInput!]): [Department!]! @authorize(policy: "ViewDepartment", apply: BEFORE_RESOLVER)
```

```
}

"Class representing a user of the application."
type User {
  "Gets or sets the id used as primary key by the database."
  id: Int!
  "Gets or sets the name the user uses in order to authenticate\nhimself."
  userName: String
  "Gets or sets the access control list associated with this\nuser."
  accessControlList: AccessControlList
  "Gets or sets the department associated with this\nuser."
  department: Department @authorize(policy: "ViewDepartment", apply: BEFORE_RESOLVER)
}

"Class containing the access control list of a specific user."
input AccessControlListFilterInput {
  and: [AccessControlListFilterInput!]
  or: [AccessControlListFilterInput!]
  "Gets or sets the acl identifier."
  id: ComparableInt32OperationFilterInput
  "Gets or sets the user associated with this list."
  user: UserFilterInput
  "Gets or sets a value indicating whether the user has permission to\nview all users."
  canViewUsers: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nview all users of his department."
  canViewDepartmentUsers: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nedit all users."
  canEditUsers: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nedit all users of his department."
  canEditDepartmentUsers: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nview all departments."
  canViewDepartments: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nview his own department."
  canViewOwnDepartment: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nedit all departments."
  canEditDepartments: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nedit his own department."
  canEditOwnDepartment: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nview all deployment plans."
  canViewDeploymentPlans: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nview all deployment plans of his department."
  canViewDepartmentDeploymentPlans: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nedit all deployment plans."
  canEditDeploymentPlans: BooleanOperationFilterInput
  "Gets or sets a value indicating whether the user has permission to\nedit all deployment plans of his department."
  canEditDepartmentDeploymentPlans: BooleanOperationFilterInput
}

"Class containing the access control list of a specific user."
input AccessControlListInput {
  "Gets or sets a value indicating whether the user has permission to\nedit all users."
  canEditUsers: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nedit all users of
```

```
his department."
  canEditDepartmentUsers: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nview all users."
  canViewUsers: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nview all users of
his department."
  canViewDepartmentUsers: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nedit all
departments."
  canEditDepartments: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nedit his own
department."
  canEditOwnDepartment: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nview all
departments."
  canViewDepartments: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nview his own
department."
  canViewOwnDepartment: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nedit all deployment
plans."
  canEditDeploymentPlans: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nedit all deployment
plans of his department."
  canEditDepartmentDeploymentPlans: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nview all deployment
plans."
  canViewDeploymentPlans: Boolean = false
  "Gets or sets a value indicating whether the user has permission to\nview all deployment
plans of his department."
  canViewDepartmentDeploymentPlans: Boolean = false
}

"Class containing the access control list of a specific user."
input AccessControlListSortInput {
  "Gets or sets the acl identifier."
  id: SortEnumType
  "Gets or sets the user associated with this list."
  user: UserSortInput
  "Gets or sets a value indicating whether the user has permission to\nview all users."
  canViewUsers: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nview all users of
his department."
  canViewDepartmentUsers: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nedit all users."
  canEditUsers: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nedit all users of
his department."
  canEditDepartmentUsers: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nview all
departments."
  canViewDepartments: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nview his own
department."
  canViewOwnDepartment: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nedit all
departments."
  canEditDepartments: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nedit his own
department."
  canEditOwnDepartment: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nview all deployment
plans."
  canViewDeploymentPlans: SortEnumType
```

```
  "Gets or sets a value indicating whether the user has permission to\nview all deployment plans of his department."
  canViewDepartmentDeploymentPlans: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nedit all deployment plans."
  canEditDeploymentPlans: SortEnumType
  "Gets or sets a value indicating whether the user has permission to\nedit all deployment plans of his department."
  canEditDepartmentDeploymentPlans: SortEnumType
}

input BooleanOperationFilterInput {
  eq: Boolean
  neq: Boolean
}

input ComparableDateTimeOperationFilterInput {
  eq: DateTime
  neq: DateTime
  in: [DateTime!]
  nin: [DateTime!]
  gt: DateTime
  ngt: DateTime
  gte: DateTime
  ngte: DateTime
  lt: DateTime
  nlt: DateTime
  lte: DateTime
  nlte: DateTime
}

input ComparableInt32OperationFilterInput {
  eq: Int
  neq: Int
  in: [Int!]
  nin: [Int!]
  gt: Int
  ngt: Int
  gte: Int
  ngte: Int
  lt: Int
  nlt: Int
  lte: Int
  nlte: Int
}

input ComparableNullableOfInt32OperationFilterInput {
  eq: Int
  neq: Int
  in: [Int]
  nin: [Int]
  gt: Int
  ngt: Int
  gte: Int
  ngte: Int
  lt: Int
  nlt: Int
  lte: Int
  nlte: Int
}

"Class representing a department."
```

```
input DepartmentFilterInput {
  and: [DepartmentFilterInput!]
  or: [DepartmentFilterInput!]
  "Gets or sets the id of the department."
  id: ComparableInt32OperationFilterInput
  "Gets or sets the name of the department."
  name: StringOperationFilterInput
  "Gets or sets list of User belonging to this\ndepartment."
  users: ListFilterInputTypeOfUserFilterInput
  "Gets or sets list of DeploymentPlan belonging to this\ndepartment."
  deploymentPlans: ListFilterInputTypeOfDeploymentPlanFilterInput
  "Gets the id of this department. Needed for\nIDepartmentComparable interface."
  departmentId: ComparableNullableOfInt32OperationFilterInput
}

"Class representing a department."
input DepartmentSortInput {
  "Gets or sets the id of the department."
  id: SortEnumType
  "Gets or sets the name of the department."
  name: SortEnumType
  "Gets the id of this department. Needed for\nIDepartmentComparable interface."
  departmentId: SortEnumType
}

"Class which represents a deployment plan object."
input DeploymentPlanFilterInput {
  and: [DeploymentPlanFilterInput!]
  or: [DeploymentPlanFilterInput!]
  "Gets or sets the id used as primary key by the database."
  id: ComparableInt32OperationFilterInput
  "Gets or sets the description which may be used to identify the\ndeployment plan."
  description: StringOperationFilterInput
  "Gets or sets the moment from which this deployment plan should be\ntreated as active."
  availableFrom: ComparableDateTimeOperationFilterInput
  "Gets or sets the moment until which the deployment plan will be\ntreated as active."
  availableUntil: ComparableDateTimeOperationFilterInput
  "Gets or sets a value indicating whether the deployment plan is in\npublished status.
Unpublished deployment plans should not be shown\nto users without administrative
permissions."
  published: BooleanOperationFilterInput
  "Gets or sets the department associated with this deployment\nplan."
  department: DepartmentFilterInput
}

"Class which represents a deployment plan object."
input DeploymentPlanSortInput {
  "Gets or sets the id used as primary key by the database."
  id: SortEnumType
  "Gets or sets the description which may be used to identify the\ndeployment plan."
  description: SortEnumType
  "Gets or sets the moment from which this deployment plan should be\ntreated as active."
  availableFrom: SortEnumType
  "Gets or sets the moment until which the deployment plan will be\ntreated as active."
  availableUntil: SortEnumType
  "Gets or sets a value indicating whether the deployment plan is in\npublished status.
Unpublished deployment plans should not be shown\nto users without administrative
permissions."
  published: SortEnumType
  "Gets or sets the department associated with this deployment\nplan."
  department: DepartmentSortInput
}
```

```
339
340 input ListFilterInputTypeOfDeploymentPlanFilterInput {
341   all: DeploymentPlanFilterInput
342   none: DeploymentPlanFilterInput
343   some: DeploymentPlanFilterInput
344   any: Boolean
345 }
346
347 input ListFilterInputTypeOfUserFilterInput {
348   all: UserFilterInput
349   none: UserFilterInput
350   some: UserFilterInput
351   any: Boolean
352 }
353
354 input StringOperationFilterInput {
355   and: [StringOperationFilterInput!]
356   or: [StringOperationFilterInput!]
357   eq: String
358   neq: String
359   contains: String
360   ncontains: String
361   in: [String]
362   nin: [String]
363   startsWith: String
364   nstartsWith: String
365   endsWith: String
366   nendsWith: String
367 }
368
369 "Class representing a user of the application."
370 input UserFilterInput {
371   and: [UserFilterInput!]
372   or: [UserFilterInput!]
373   "Gets or sets the id used as primary key by the database."
374   id: ComparableInt32OperationFilterInput
375   "Gets or sets the name the user uses in order to authenticate\nhimself."
376   userName: StringOperationFilterInput
377   "Gets or sets the access control list associated with this\nuser."
378   accessControlList: AccessControlListFilterInput
379   "Gets or sets the department associated with this\nuser."
380   department: DepartmentFilterInput
381 }
382
383 "Class representing a user of the application."
384 input UserSortInput {
385   "Gets or sets the id used as primary key by the database."
386   id: SortEnumType
387   "Gets or sets the name the user uses in order to authenticate\nhimself."
388   userName: SortEnumType
389   "Gets or sets the access control list associated with this\nuser."
390   accessControlList: AccessControlListSortInput
391   "Gets or sets the department associated with this\nuser."
392   department: DepartmentSortInput
393 }
394
395 enum ApplyPolicy {
396   BEFORE_RESOLVER
397   AFTER_RESOLVER
398 }
399
```

```
enum SortEnumType {
  ASC
  DESC
}

directive @authorize("The name of the authorization policy that determines access to the annotated resource." policy: String "Roles that are allowed to access the annotated resource." roles: [String!] "Defines when when the resolver shall be executed.By default the resolver is executed after the policy has determined that the current user is allowed to access the field." apply: ApplyPolicy! = BEFORE_RESOLVER) repeatable on SCHEMA | OBJECT | FIELD_DEFINITION

"The `@defer` directive may be provided for fragment spreads and inline fragments to inform the executor to delay the execution of the current fragment to indicate deprioritization of the current fragment. A query with `@defer` directive will cause the request to potentially return multiple responses, where non-deferred data is delivered in the initial response and data deferred is delivered in a subsequent response. `@include` and `@skip` take precedence over `@defer`."
directive @defer("If this argument label has a value other than null, it will be passed on to the result of this defer directive. This label is intended to give client applications a way to identify to which fragment a deferred result belongs to." label: String "Deferred when true." if: Boolean) on FRAGMENT_SPREAD | INLINE_FRAGMENT

"The `@specifiedBy` directive is used within the type system definition language to provide a URL for specifying the behavior of custom scalar definitions."
directive @specifiedBy("The specifiedBy URL points to a human-readable specification. This field will only read a result for scalar types." url: String!) on SCALAR

"The `@stream` directive may be provided for a field of `List` type so that the backend can leverage technology such as asynchronous iterators to provide a partial list in the initial response, and additional list items in subsequent responses. `@include` and `@skip` take precedence over `@stream`."
directive @stream("If this argument label has a value other than null, it will be passed on to the result of this stream directive. This label is intended to give client applications a way to identify to which fragment a streamed result belongs to." label: String "The initial elements that shall be send down to the consumer." initialCount: Int! "Streamed when true." if: Boolean!) on FIELD

"The `DateTime` scalar represents an ISO-8601 compliant date time type."
scalar DateTime @specifiedBy(url: "https:\/\/www.graphql-scalars.com\/date-time")

"The `Upload` scalar type represents a file upload."
scalar Upload

scalar Url
```