

# Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem

IP6 - Bachelor Thesis

24. März 2022

Studenten      Joshua Villing

Fachbetreuer    Daniel Jossen

Auftraggeber    Daniel Jossen

Studiengang    Informatik

Hochschule    Hochschule für Technik

## Management Summary

”Ärzte und Zahnärzte haben den Anspruch in Ihren Praxen ein Rufsystem einzusetzen. Dieses Rufsystem ermöglicht, dass der behandelnde Arzt über einen Knopfdruck Hilfe anfordern oder Behandlungsmaterial bestellen kann. Heute kommerziell erhältliche Rufsysteme beruhen meistens auf proprietären Standards und veralteten Technologien.”[1]. Mit dem Vorgängerprojekt ”IP5 Cloudbasiertes Praxisrufsystem” wurde ein modernes Rufsystem ”Praxisruf” mit reduziertem Funktionsumfang umgesetzt. Die Problemstellung für dieses Projekt besteht darin, Praxisruf zu erweitern und die fehlenden Funktionen ”Gegensprechanlage” und ”Vorlesen von Benachrichtigungen” zu ergänzen.

Das erweiterte Praxisrufsystem hat zwei Hauptfunktionen. Es kann als Gegensprechanlage verwendet werden und bietet die Möglichkeit vorkonfigurierte Benachrichtigungen zu versenden. Das Rufsystem wird von Endbenutzern über iPads bedient. Die dazu entwickelte App ermöglicht es, Sprachverbindungen von einem Gerät zu einem oder mehreren Gesprächspartnern aufzubauen und Benachrichtigungen zu versenden. Der Inhalt von empfangenen Benachrichtigungen wird bei entsprechender Konfiguration automatisch vorgelesen. Welche Sprachverbindungen und Benachrichtigungen zur Verfügung stehen, wird von Administratoren über eine Weboberfläche konfiguriert.

Nach Projektabschluss stehen drei Optionen für das weitere Vorgehen offen:

### **1. Produktiver Einsatz**

Mit den Erweiterungen, die in diesem Projekt umgesetzt wurden, erfüllt Praxisruf die Mindestanforderungen an ein modernes Praxisrufsystem. Dementsprechend könnte das System bereits heute als produktives Rufsystem in einem Praxisumfeld eingesetzt werden. Praxisruf wurde allerdings noch nicht in einem produktiven Umfeld getestet. Dementsprechend konnte nicht verifiziert werden, dass das System mit der Last, die in einem produktiven Umfeld entsteht, umgehen kann. Weiter weist das System in den Bereichen Stabilität und Benutzerfreundlichkeit Verbesserungspotential auf. Aufgrund dieser Einschränkung wird davon abgeraten, Praxisruf produktiv einzusetzen.

### **2. Weiterentwicklung mit Pilotbetrieb**

Um Praxisruf in einem produktiven Umfeld zu testen, könnte es als Pilotprojekt in einer Praxis eingesetzt werden. In diesem Pilotprojekt könnte das Verhalten des Systems in einem realistischen Umfeld beobachtet werden. Es könnten Rückmeldung von Praxismitarbeitenden gesammelt und die Performance des Systems gemessen werden. Gleichzeitig kann das System weiterentwickelt werden. Bekannte Lücken können behoben werden. Dabei können die Erkenntnisse aus dem Pilotbetrieb berücksichtigt werden.

### **3. Weiterentwicklung für kommerzielle Nutzung**

Praxisruf ist heute darauf ausgelegt, in genau einer Praxis eingesetzt zu werden. Um es als kommerziell erhältliches System anbieten zu können, muss es mandantenfähig werden. Es muss möglich sein, Konfigurationen mehrerer Praxen zu verwalten, wobei diese strikt voneinander getrennt bleiben müssen. Zu diesem Zweck wird empfohlen die Berechtigungsprüfung in Praxisruf zu erweitern. Es sollte ein detailliertes Rollenkonzept entwickelt und ein externer Identity-Provider angebunden werden.

### **Empfehlung**

Es wird empfohlen mit der Option zwei ”Test in Pilotbetrieb” fortzufahren. Durch Pilotbetrieb in einem realistischen Umfeld, kann garantiert werden, dass Praxisruf den Qualitätsansprüchen der Zielgruppe entspricht. Gleichzeitig kann das System weiterentwickelt und bekanntes Optimierungspotential realisiert werden. Sollte die Absicht verfolgt werden, Praxisruf als kommerzielles System anzubieten, wird zudem empfohlen die Option drei ”Weiterentwicklung für kommerzielle Nutzung” zu verfolgen. Die Erweiterung um Mandantenfähigkeit und Ergänzungen basierend auf Erkenntnissen aus einem Pilotbetrieb sind für die kommerzielle Nutzung von Praxisruf unerlässlich.

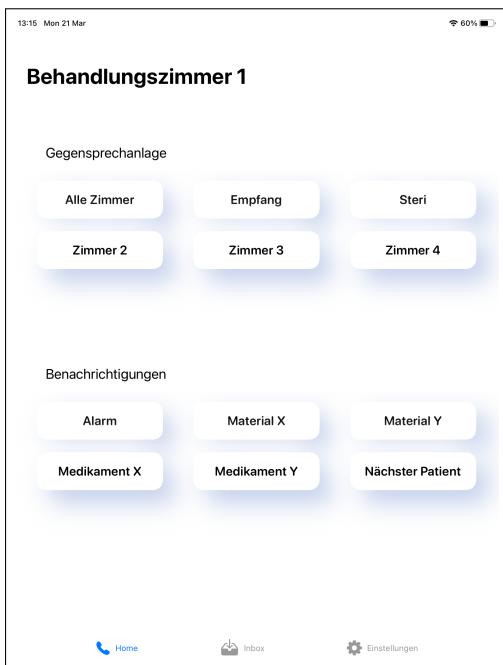
## Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Vorgehensweise</b>	<b>4</b>
2.1 Projektplan . . . . .	4
2.2 Meilensteine . . . . .	5
<b>3 Anforderungen</b>	<b>6</b>
3.1 User Stories . . . . .	6
3.2 Features . . . . .	7
<b>4 Ausgangslage</b>	<b>8</b>
4.1 Bestehende Rufsysteme . . . . .	8
4.2 Vorarbeit Cloudbasiertes Praxisrufsystem . . . . .	8
<b>5 Technologieentscheid</b>	<b>9</b>
5.1 Native iOS Applikation . . . . .	9
5.2 Sprachsynthese . . . . .	12
5.3 Gegensprechanlage . . . . .	14
<b>6 Peer-To-Peer Sprachübertragung</b>	<b>16</b>
6.1 Verbindungsprotokolle . . . . .	16
6.2 Verbindungsaufbau . . . . .	16
6.3 Signaling . . . . .	17
6.4 Sicherheit . . . . .	17
6.5 Unicast, Multi-Cast, Broadcast . . . . .	18
<b>7 Konzept</b>	<b>19</b>
7.1 Systemarchitektur . . . . .	19
7.2 Native iOS Applikation . . . . .	23
7.3 Sprachsynthese . . . . .	30
7.4 Gegensprechanlage . . . . .	36
<b>8 Umsetzung</b>	<b>48</b>
8.1 Resultate . . . . .	48
8.2 Tests . . . . .	53
8.3 Fazit . . . . .	57

<b>9 Schluss</b>	<b>59</b>
<b>Literaturverzeichnis</b>	<b>61</b>
<b>Abbildungsverzeichnis</b>	<b>64</b>
<b>A Aufgabenstellung</b>	<b>66</b>
<b>B Quellcode</b>	<b>67</b>
<b>C Features und Testszenarien</b>	<b>68</b>
<b>D Installationsanleitung</b>	<b>72</b>
<b>E Benutzerhandbuch</b>	<b>78</b>
<b>F Ehrlichkeitserklärung</b>	<b>81</b>

# 1 Einleitung

„Ärzte und Zahnärzte haben den Anspruch in Ihren Praxen ein Rufsystem einzusetzen. Dieses Rufsystem ermöglicht, dass der behandelnde Arzt über einen Knopfdruck Hilfe anfordern oder Behandlungsmaterial bestellen kann.“[1] Mit diesem Projekt wurde ein cloudbasiertes Praxisrufsystem realisiert. Dazu wurde das im Vorgängerprojekt „IP5 Cloudbasiertes Praxisrufsystem“ umgesetzte Praxisrufsystem erweitert. Das erweiterte System ermöglicht es Sprachverbindungen zwischen Teilnehmern aufzubauen und Benachrichtigungen zu versenden. Der Inhalt von empfangenen Benachrichtigung kann automatisch vorgelesen werden. Als Benutzeroberfläche dient eine native iOS Applikation. Folgende Abbildungen zeigen die Startseite dieser Applikation (Abbildung 1.1) sowie die Ansicht während eines aktiven Gruppenanrufs (Abbildung 1.2).



**Abbildung 1.1:** Hauptansicht



**Abbildung 1.2:** Aktiver Anruf

Auf der Startseite der Applikation können per Knopfdruck Benachrichtigungen versendet und Sprachverbindungen gestartet werden. Empfangene Benachrichtigungen werden als Push-Benachrichtigung angezeigt und in einer Inbox gesammelt. Bei entsprechender Konfiguration wird zudem der Inhalt empfangener Benachrichtigungen vorgelesen. Sprachverbindungen können zwischen zwei oder mehr Teilnehmern aufgebaut werden. Das System unterstützt sowohl private Gespräche als 1:1 Verbindungen wie auch Gruppenunterhaltungen als 1:n Verbindungen.

Welche Buttons und damit welche Benachrichtigungen und Sprachverbindungen zur Verfügung stehen, wird durch Administratoren konfiguriert. Die Konfiguration von Buttons für Sprachverbindungen beinhaltet, mit welchen Empfängern eine Verbindung aufgebaut wird. Die Konfiguration von Buttons für Benachrichtigungen definiert den Inhalt der Benachrichtigung, welche Empfänger sie erhalten und ob die Benachrichtigung vorgelesen wird. Die Verwaltung dieser Konfiguration kann durch eine Weboberfläche vorgenommen werden. Abbildung 1.3 zeigt die Übersicht verfügbarer Benachrichtigung in dieser Weboberfläche.

Die Grundlage für das umgesetzte Praxisrufsystem wurde im Rahmen der Projektarbeit „IP5 Cloudbasiertes Praxisrufsystem“ erarbeitet. Im Rahmen des Vorgängerprojektes wurde ein Praxisrufsystem mit eingeschränktem Funktionsumfang realisiert. Die in diesem Projekt entwickelte Lösung ist eine Erweiterung des bestehenden Systems. Das zuvor umgesetzte System unterstützte das Versenden und Empfangen

The screenshot shows a web-based application interface titled "Notification Types". On the left, there is a sidebar with navigation links: "Users", "Clients", "Notification Types" (which is currently selected), "Configurations", and "Calltypes". The main content area displays a table with the following data:

	Title	Body	Description	Text to speech
<input type="checkbox"/>	Display text	Material	Material wird benötigt	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Material bringen	Nächster Patient	Nächster Patient bereit	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Nächster Patient			<input checked="" type="checkbox"/>
<input type="checkbox"/>	Alarm	Schlimme Sache	Alarm	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Material bereit	Material bereit	Material ist sterilisiert	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Verpasster Anruf	Missed Call		<input checked="" type="checkbox"/>

At the bottom right of the table, it says "Rows per page: 10 - 1-5 of 5".

Abbildung 1.3: Praxisruf Startseite

von Benachrichtigungen. Sprachsynthese für Benachrichtigungen und Sprachverbindungen für eine Gegensprechanlage konnten im Rahmen des Vorgängerprojektes nicht umgesetzt werden [2]. Die zentrale Aufgabenstellung für dieses Projekt ist es, das System zu erweitern so, dass Benachrichtigungen vorgelesen und Sprachverbindungen aufgebaut werden können. Die Bedienung des Praxisrufsystems muss weiterhin über eine mobile Applikation möglich sein. Dazu soll eine neue, native iOS Applikation entwickelt werden. Diese ersetzt die bestehende App und muss alle bestehenden Funktionen unterstützen. Sie muss zudem Sprachverbindungen zu anderen Teilnehmern aufbauen und den Inhalt von Benachrichtigungen vorlesen können.

Wie in der Aufgabenstellung beschrieben, hat eine Marktanalyse im Vorfeld dieses Projektes gezeigt, dass bestehende kommerzielle Praxisrufsysteme vorwiegend veraltete Kommunikationstechnologien einsetzen. Diese Systeme seien deshalb aufwändig zu installieren und skalieren schlecht. Sie können weiter nicht einfach in ein TCP/IP-Netzwerk eingebunden oder über externe APIs angesteuert werden [1]. Das im Rahmen des Vorgängerprojektes umgesetzte System löst diese Probleme teilweise. Mit der Serverkomponente "Cloudservice" bietet das System einen zentralen Dienst, welcher über eine Http-Schnittstelle ansprechbar ist. Diese ermöglicht die Verwaltung der Systemkonfiguration und leitet Benachrichtigung anhand der Konfiguration an relevante Empfänger weiter. Dadurch ist es einfacher möglich, das System in Netzwerke einzubinden, zu skalieren und neue Endgeräte anzubinden. Das Vorgängersystem unterstützt aber noch nicht alle Funktionen, die ein Praxisrufsystem bieten muss. Die meisten kommerziell erhältlichen Lösungen können als Gegensprechanlage verwendet werden [1]. Ein konkurrenzfähiges Praxisrufsystem muss deshalb als Gegensprechanlage verwendet werden können. Mit der Integration von Sprachsynthese für Benachrichtigungen kann sich das System weiter von bestehenden Lösungen absetzen. Ein weiterer Schwachpunkt des Vorgängerprojektes ist die darin entwickelte mobile Applikation. Diese wurde mit einer geteilten Codebasis für iOS und Android entwickelt. Im Fazit des Vorgängerprojekts wurde festgehalten, dass diese Applikation neu als native Applikation entwickelt werden sollte. Dadurch könnte effizienter Betrieb, Wartung sowie Hardware- und Betriebssystemkompatibilität langfristig gewährleistet werden [2]. Deshalb ist auch die Neuentwicklung und Erweiterung der mobile App als native Applikation zentraler Bestandteil der Aufgabenstellung.

Das umgesetzte System besteht aus drei Applikationen. Der zentrale Cloudservice dient zur Verwaltung der Konfiguration und dem Vermitteln von Nachrichten zwischen Endgeräten. Das Admin UI bietet eine Weboberfläche, mit der die Konfiguration des Cloudservice verwaltet werden kann. Mit dem Mobile Client bietet das System neu eine native iOS App über welche Sprachverbindungen aufgebaut und Benachrichtigungen versendet/empfangen werden können. Für das Versenden von Benachrichtigungen senden Mobile Clients HTTP Anfragen an den Cloudservice. Der Cloudservice findet in der Konfiguration alle relevanten Empfänger und leitet die Benachrichtigung an diese weiter. Für die Zustellung von dem Cloudservice an Mobile Clients wird Firebase Cloud Messaging verwendet.

Das Vorlesen von Benachrichtigungen wurde mit einer Anbindung an Amazon Polly im Cloudservice implementiert. Bei Amazon Polly handelt es sich um einen Dienst zur Sprachsynthese von Amazon Webservices [3]. Der Cloudservice bietet eine Http-Schnittstelle, über welche Clients Sprachdaten für

Benachrichtigungen beziehen können. Durch diese Lösung müssen die Endgeräte die Anbindung an den Provider für Sprachsynthese nicht selbst implementieren. Dies bietet insbesondere die Vorteile, dass der Provider leicht ausgetauscht werden kann und dass die Anbindung von zukünftigen Plattformen übernommen werden kann.

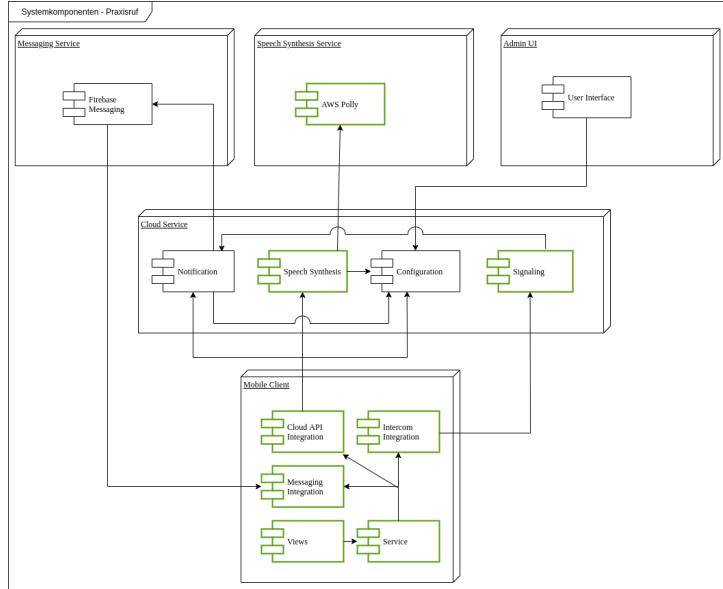


Abbildung 1.4: Systemarchitektur Praxisruf

Abbildung 1.4 gibt einen Überblick über die Komponenten des umgesetzten Systems. Dabei sind Module, die im Rahmen dieses Projektes entwickelt oder neu integriert werden, grün umrandet. Pfeile in der Grafik stellen den Informationsfluss zwischen Komponenten dar.

Sprachverbindungen zwischen Mobile Clients werden als Peer-To-Peer Verbindungen aufgebaut. Diese Verbindungen wurden mit der Technologie WebRTC umgesetzt. WebRTC basiert auf einem offenen Standard, welcher Echtzeitkommunikation für mobile Applikationen und Browser Applikationen ermöglicht [4]. Damit Peer-To-Peer Verbindungen zwischen Endgeräten aufgebaut werden können, müssen diese Signalmeldungen austauschen können. Um dies zu ermöglichen, wurde der Cloudservice um ein Modul "Signaling" erweitert. Dieses Modul bietet eine WebSocket-Schnittstelle, über welche synchrone Meldungen ausgetauscht werden können. Jedes Endgerät, das für Sprachverbindungen verfügbar ist, baut bei der Anmeldung im System eine langlebige Verbindung zum Signaling-Modul auf. Über diese Verbindung sendet und empfängt es alle Signalmeldungen, die für den Verbindungsauflauf notwendig sind.

Im folgenden Bericht werden die erarbeiteten Konzepte und Resultate detailliert beschrieben. Zunächst werden Vorgehensweise, Projektplan und die Organisation des Projekts vorgestellt. Anschliessend werden die Anforderungen, welche zu Beginn des Projekts definiert wurden, beschrieben. Es folgen eine Zusammenfassung der Ausgangslage und eine Technologie Evaluation für Sprachsynthese und Gegensprechsanlage. Dabei werden mögliche Optionen beschrieben und anschliessend begründete Entscheidungen getroffen. Das darauffolgende Kapitel gibt eine Einführung in die technischen Grundlagen und Protokolle für WebRTC. Im anschliessenden Hauptteil wird das detaillierte technische Konzept für Funktionsweise und Architektur des Systems dokumentiert. Es wird beschrieben, wie die Funktionen Gegensprechsanlage und Sprachsynthese umgesetzt werden. Dabei wird insbesondere darauf eingegangen, wie die bestehende und neue Funktionen in einer nativen iOS Applikation integriert werden. Weiter werden die notwendigen Abläufe, Kommunikationskanäle und Datenmodelle beschrieben. Nach dem Konzept werden die Resultate zusammengefasst und die umgesetzten Ansichten des nativen iOS App abgebildet. Am Ende der Arbeit stehen ein Fazit und Schlusswort mit Empfehlungen für das weitere Vorgehen.

## 2 Vorgehensweise

Dieses Kapitel beschreibt die Planung des Projekts, welche zu Projektbeginn definiert wurde. Es wurde ein Projektplan erarbeitet und Meilensteine zur Fortschrittskontrolle erfasst.

### 2.1 Projektplan

Der Projektplan wurde zu Beginn der Projektarbeit definiert und wie in Abbildung 2.1 dargestellt festgehalten. Folgendes Kapitel beschreibt den damit definierten Ablauf.

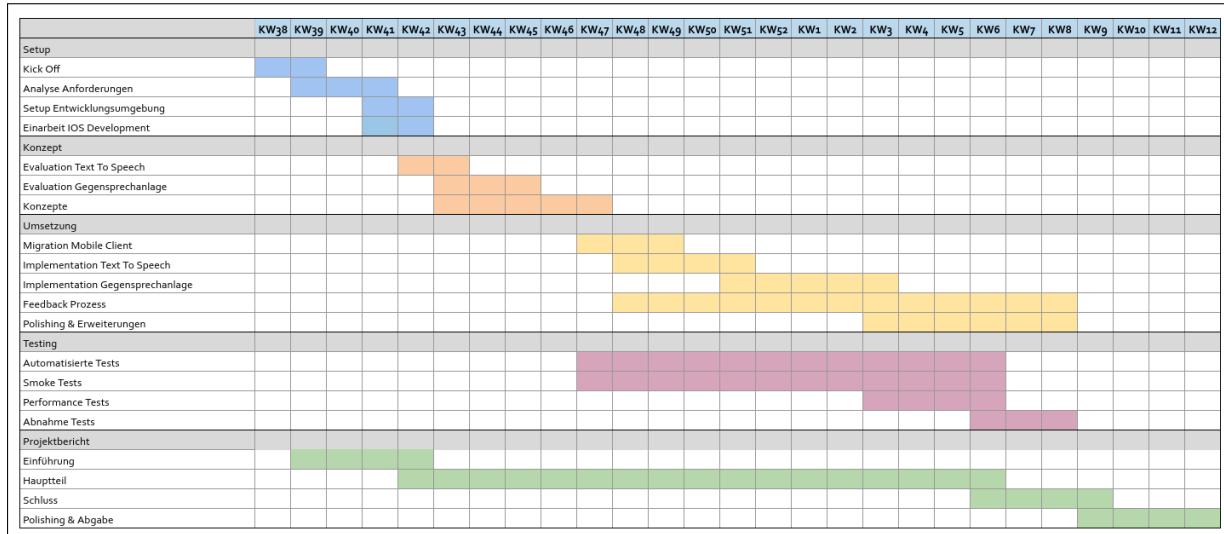


Abbildung 2.1: Projektplan

Das Projekt beginnt mit einer Setup-Phase. Anforderungen und Umfang des Projektes werden zusammen mit dem Auftraggeber erarbeitet und dokumentiert. Es wird ein Projektplan erstellt und die grobe Struktur des Projektberichts definiert. Infrastruktur und Codebasis des Vorgängerprojektes werden übernommen und wo nötig für dieses Projekt angepasst. Für die Entwicklung einer nativen iOS App muss eine Entwicklungsumgebung inklusive Hardware und Testgeräten aufgesetzt werden. Während dieser Zeit findet bereits eine Einarbeitung in native iOS Entwicklung statt. So können erste Erfahrungen gesammelt werden und die Vollständigkeit der Entwicklungsumgebung validiert werden. Nach der Setup-Phase beginnt die Konzept-Phase. Dies beinhaltet Evaluation von Technologien sowie Definition von Grundprozessen und Datenstrukturen. Die erstellten Konzepte werden mit einfachen gehaltenen Proof of Concepts validiert. Alle erarbeiteten Konzepte werden dokumentiert. Es wird beschrieben, wie die bestehende Systemarchitektur erweitert wird, um die neuen Funktionen zu integrieren. Dabei müssen insbesondere Skalierbarkeit, Erweiterbarkeit und Integrierbarkeit des Systems bewahrt werden. Die Umsetzung des Systems findet schliesslich in der dritten Phase statt. Die erarbeiteten Konzepte werden umgesetzt und in das Praxisrufsystem integriert. Bei der Umsetzung wird als Erstes der bestehende Mobile Client als native iOS Applikation neu entwickelt. Danach werden Sprachsynthese und Gegensprechanlage implementiert. Die erarbeiteten Konzepte werden dabei laufend verfeinert und wo nötig angepasst. Während der Umsetzung wird der aktuelle Stand regelmässig mit dem Kunden besprochen. Der Kunde kann so frühzeitig Rückmeldung geben. Gewünschte Anpassungen und Erweiterungen können frühzeitig eingeplant werden. Parallel zur Umsetzung läuft eine Testing-Phase. Diese beinhaltet das Schreiben von automatisierten Tests und regelmässigen manuellen Tests des aktuellen Stands. Zum Ende der Umsetzungsphase wird schliesslich Benutzertests mit dem Kunden durchgeführt. Die vierte Phase bildet den Abschluss des Projekts. Es sollen keine weiteren Änderungen an der Applikation vorgenommen werden. Der Projektbericht wird fertiggestellt und das Projekt wird für die Abgabe vorbereitet.

## 2.2 Meilensteine

In der Anfangsphase des Projektes wurden folgende Meilensteine definiert:

<b>Id</b>	<b>Beschreibung</b>
M01	<b>Initiale Anforderungsanalyse</b> Die Anforderungen an das Projekt aus der Aufgabenstellungen sind in User Stories dokumentiert.
M02	<b>Einarbeit und Setup IOS Umgebung</b> Projektteilnehmende sind mit groben Konzepten der IOS Entwicklung vertraut. Die Entwicklungs-umgebung ist bereit für die Umsetzung.
M03	<b>Evaluation Technologien</b> Die Evaluation der Technologien für Sprachsynthese und Gegensprechanlage ist abgeschlossen.
M04	<b>Konzepte</b> Die grundlegenden Konzepte für Systemarchitektur, native iOS Applikation, Gegensprechanlage und Sprachsynthese sind definiert.
M05	<b>Migration bestehender Funktionalität</b> Die Funktionen die im Mobile Client der Projektarbeit "IP5 Cloudbasiertes Praxisrufsystem" umgesetzt wurden, stehen in der neuen iOS Applikation zur Verfügung.
M06	<b>Umsetzung Sprachsynthese</b> Die Anforderung zur Funktion Sprachsynthese sind umgesetzt.
M07	<b>Umsetzung Gegensprechanlage 1:1</b> Die Anforderung zur Funktion Gegensprechanlage sind für 1:1 Verbindungen umgesetzt.
M08	<b>Umsetzung Gegensprechanlage 1:n</b> Die Anforderung zur Funktion Gegensprechanlage sind für 1:n Verbindungen umgesetzt.
M09	<b>Polishing und Erweiterungen</b> Die Applikation wurde eingehend getestet. Bekannte Fehler sind behoben oder dokumentiert. Gewünschte Anpassungen und Erweiterungen sind umgesetzt oder dokumentiert.
M10	<b>Abschluss und Abgabe</b> Projektbericht und Quellcode sind fertiggestellt. Projektcode und Quellcode des umgesetzten Systems wurden pünktlich abgegeben.

## 3 Anforderungen

Dieses Kapitel beschreibt die Anforderungen an das Praxisrufsystem, welche zu Beginn des Projektes erarbeitet wurden. Die Anforderungen werden aus Sicht der Stakeholder festgehalten. Dazu werden zunächst Stakeholdergruppen identifiziert und anschliessend pro Gruppe User Stories definiert. Jede User Story beschreibt eine Anforderung, welche eine Stakeholdergruppe an das System stellt. Anhand der User Stories werden anschliessend Features definiert, welche User Stories unter einem gemeinsamen Titel zusammenfassen.

### 3.1 User Stories

Im folgenden Kapitel werden die Anforderungen aus Sicht der Stakeholder dokumentiert. Im Vorgängerprojekt wurden die drei Stakeholdergruppen Praxismitarbeitende, Praxisverantwortliche und Auftraggeber identifiziert. Die Gruppe Praxismitarbeitende verwendet das Praxisrufsystem, um in der Praxis zu kommunizieren. Praxisverantwortliche sind dafür zuständig, die Konfiguration des Praxisrufsystems zu erstellen und zu verwalten. Als dritte Stakeholdergruppe hat der Auftraggeber ein Interesse daran, dass gewisse Rahmenbedingungen gesetzt und eingehalten werden [2]. Die Stakeholdergruppen des Vorgängerprojektes werden für dieses Projekt übernommen. Im Folgenden werden die erarbeiteten Anforderungen pro Stakeholdergruppe tabellarisch aufgelistet.

#### 3.1.1 Praxismitarbeitende

<b>Id</b>	<b>Anforderung</b>
U01	Als Praxismitarbeiter/in möchte ich alle bestehenden Funktionen von Praxisruf weiterhin verwenden können, damit mir diese weiterhin die Arbeit erleichtern.
U02	Als Praxismitarbeiter/in möchte ich, dass eingehende Benachrichtigungen vorgelesen werden, damit ich den Inhalt der Benachrichtigung kenne, ohne meine Aufmerksamkeit auf den Bildschirm zu richten.
U03	Als Praxismitarbeiter/in möchte ich, das Vorlesen von Benachrichtigungen deaktivieren können, damit ich bei der Arbeit nicht unnötig gestört werde.
U04	Als Praxismitarbeiter/in möchte ich, per Button eine Sprachverbindung zu einem anderen Praxiszimmer aufbauen können, damit ich mich mit einer anderen Person absprechen kann.
U05	Als Praxismitarbeiter/in möchte ich, per Button eine Sprachverbindung zu mehreren anderen Praxiszimmern aufbauen können damit, ich mich mit mehreren anderen Personen absprechen kann.
U06	Als Praxismitarbeiter/in möchte ich über geöffnete Sprachverbindungen in Echtzeit kommunizieren können, damit Praxisruf die Funktion einer Gegensprechanlage erfüllt.
U07	Als Praxismitarbeiter/in möchte ich nur Buttons für Sprachverbindungen sehen, die für mich relevant sind, damit ich das System einfacher bedienen kann.
U08	Als Praxismitarbeiter/in möchte ich benachrichtigt werden, wenn ein anderes Zimmer eine Sprachverbindung öffnet, damit ich auf die Anfrage antworten kann.
U09	Als Praxismitarbeiter/in möchte ich den Verlauf empfangener und verpasster Sprachverbindungen nachvollziehen können, damit ich mich zurückmelden kann.
U10	Als Praxismitarbeiter/in möchte ich, dass eingehende Sprachverbindungen aus anderen Praxiszimmern automatisch geöffnet werden, damit das Gespräch möglichst schnell beginnen kann.
U11	Als Praxismitarbeiter/in möchte ich, direkte Sprachverbindungen aus anderen Praxiszimmern trennen können, damit ich ein Gespräch beenden kann.
U12	Als Praxismitarbeiter/in möchte ich, aus Sprachverbindungen zu mehreren Praxiszimmern (Gruppenunterhaltungen) austreten können, damit ich nicht unnötig bei der Arbeit gestört werde.

### 3.1.2 Praxisverantwortliche

<b>Id</b>	<b>Anforderung</b>
U13	Als Praxisverantwortliche/-r möchte ich konfigurieren können, welche Benachrichtigungen vorgelesen werden, damit nur relevante Benachrichtigungen vorgelesen werden.
U14	Als Praxisverantwortliche/-r möchte ich konfigurieren können, aus welchen Zimmern Sprachverbindungen zu welchen anderen Zimmern aufgebaut werden können, damit Praxismitarbeitende das System effizient bedienen können.
U15	Als Praxisverantwortliche/-r möchte ich Benachrichtigungen, Geräte und Benutzer wie zuvor konfigurieren können, damit ich das System weiterhin für meine Praxis konfigurieren kann.

### 3.1.3 Auftraggeber

<b>Id</b>	<b>Anforderung</b>
U16	Als Auftraggeber möchte ich bestehende Betriebsinfrastruktur übernehmen, um von der bereits geleisteten Arbeit profitieren zu können.
U17	Als Auftraggeber möchte ich, dass die bestehenden Komponenten des Systems wo immer möglich weiter verwendet werden, um von der bereits geleisteten Arbeit profitieren zu können.
U18	Als Auftraggeber möchte ich, dass der bestehende Mobile Client als native iOS Applikation neu entwickelt wird, um Weiterentwicklung, Betrieb und Gerätekompatibilität langfristig zu gewährleisten.
U19	Als Auftraggeber möchte ich, dass für Betrieb eigener Services und Bezug externer Dienstleistungen Amazon Webservices verwendet wird, damit ich von bestehender Infrastruktur und Erfahrung profitieren kann.

## 3.2 Features

Die in Kapitel 3.1 beschriebenen User Stories werden in drei Features gruppiert. Diese drei Features bilden Grundlage für Aufteilung der nachfolgenden Kapitel Konzept und Technologie Evaluation. Es wurde die folgenden drei Features definiert:

<b>Id</b>	<b>Feature</b>
F01	Native iOS Applikation
F02	Sprachsynthese
F03	Gegensprechanlage

Das Feature F01 "Native iOS Applikation" bildet die Grundlage für die Neuentwicklung der mobilen Applikation. Es beinhaltet den Aufbau der neuen Applikation und die Migration bestehender Funktionalität aus dem Vorgängerprojekt. Das Feature F02 "Sprachsynthese" baut auf F01 auf und fügt die Sprachsynthese für empfangene Benachrichtigungen hinzu. Mit dem Feature F03 "Gegensprechanlage" wird schliesslich das Kernstück der neuen Funktionalität umgesetzt. Dieses Feature beinhaltet Integration von Peer-To-Peer Sprachverbindungen in das System. Nach der Umsetzung dieses Features kann das Praxisrufsystem als Gegensprechanlage verwendet werden. Die korrekte Umsetzung aller Anforderungen wird durch Funktionstests sichergestellt. Dazu werden Testszenarien definiert, welche Ausgangslage, Testschritte und die erwarteten Resultate definieren. Die Szenarien sind in Anhang C aufgeführt. Die Resultate der letzten Ausführung der Testszenarien sind im Kapitel 8.2 festgehalten.

## 4 Ausgangslage

Dieses Kapitel beschreibt die Ausgangslage zu Beginn dieser Projektarbeit. Es wird eine Übersicht zu bestehenden Lösungen und den Problemen, welche diese haben gegeben. Dabei wird auch beschrieben, wie diese Probleme in einem cloudbasierten Praxisrufsystem gelöst werden. Weiter wird ein Überblick zu dem im Vorgängerprojekt erarbeiteten cloudbasierten Praxisrufsystem gegeben.

### 4.1 Bestehende Rufsysteme

Im Vorfeld dieses Projektes wurde eine Marktanalyse zu kommerziell erhältlichen Rufsystemen durchgeführt. Die Resultate dieser Analyse sind in der Aufgabenstellung dieses Projektes zusammengefasst: Die meisten kommerziell erhältlichen Rufsysteme basieren auf proprietären Standards und setzen veraltete Funktechnologien ein. Diese Systeme sind weder über TCP/IP-Netzwerke integrierbar, noch über externe APIs ansteuerbar [1]. Bestehende Systeme sind deshalb oft aufwändig zu installieren und konfigurieren. Erweiterung und Veränderungen am System können nur mit grossem Aufwand vorgenommen werden.

Ein cloudbasiertes Praxisrufsystem kann diese Probleme lösen. Durch die Verwaltung der Konfiguration in Clouddiensten können Schnittstellen zur Integration von Endgeräten geboten werden. Weiter können Schnittstellen geschaffen werden, welche die Kommunikation anhand dieser Konfiguration vermitteln. Endgeräte können diese Schnittstellen verwenden, um ihre Konfiguration zu beziehen und mit anderen Endgeräten zu kommunizieren. Ein solches Rufsystem kann einfach verändert und erweitert werden. Die Konfiguration kann über die Schnittstellen der Clouddienste verwaltet werden. Neue Endgeräte können über diese Schnittstellen einfach in das System integriert werden. Clouddienste können unabhängig von den Endgeräten skaliert werden und über die Schnittstellen in bestehende Netzwerke integriert werden.

### 4.2 Vorarbeit Cloudbasiertes Praxisrufsystem

Mit dem Vorgängerprojekt "Cloudbasiertes Praxisrufsystem" wurde ein cloudbasiertes Rufsystem mit dem Namen Praxisruf entwickelt. Dieses ermöglicht es, vorkonfigurierte Benachrichtigungen zu versenden und empfangen. Das System unterstützt allerdings noch keine Sprachverbindungen oder Sprachausgabe.

Praxisruf umfasst eine zentrale Serverkomponente, eine Weboberfläche und eine mobile Applikation für iOS und Android Tablets. Die mobile Applikation dient als Benutzeroberfläche für Praxismitarbeitende. Sie ermöglicht es vorkonfigurierte Benachrichtigung zu senden und empfangen. Die Konfiguration des Systems wird mit der Serverkomponente "Cloudservice" verwaltet. Der Cloudservice bietet dazu eine API an, über welche Konfigurationen erfasst, gelesen und bearbeitet werden können. Die Weboberfläche "Admin UI" bietet eine Benutzeroberfläche, um Konfigurationen zu verwalten. Das Admin UI verwendet dazu die API des Cloudservice. Die API des Cloudservice wird weiter von der mobilen Applikation verwendet, um die Konfiguration zu laden und Benachrichtigungen zu versenden.

Das Senden und Empfangen von Benachrichtigungen in Praxisruf wird durch Firebase Cloud Messaging (FCM) ermöglicht. Sowohl in der mobilen Applikation als auch im Cloudservice ist eine Anbindung an FCM implementiert. Dabei wird die Anbindung auf der Seite der App ausschliesslich zum Empfangen von Meldungen verwendet. Das Versenden von Benachrichtigungen wird an den Cloudservice delegiert. Dazu wird die API des Cloudservice verwendet. Der Cloudservice wertet die Konfiguration aus, um relevante Empfänger zu identifizieren. Anschliessend benutzt er die FCM-Anbindung, um die Benachrichtigung an alle relevanten Empfänger zuzustellen.

Sämtliche Infrastruktur für Praxisruf ist mit Amazon Webservices (AWS) eingerichtet. Der Java-Applikation Cloudservice wird mit einer AWS Elastic Beanstalk Instanz betrieben. Die Weboberfläche Admin UI wird mit AWS Amplify betrieben [2].

## 5 Technologieentscheid

In diesem Kapitel wird entschieden, mit welchen Technologien und Frameworks die Anforderungen an das Projekt umgesetzt werden. Dies beinhaltet die Neuentwicklung des bestehenden Mobile Clients als native iOS Applikation, sowie die Implementation der Features Sprachsynthese und Gegensprechanlage.

### 5.1 Native iOS Applikation

Mit dem Projekt "IP5 Cloudbasiertes Praxisrufsystem" wurde eine mobile Applikation entwickelt, die als Endpunkt in einem Praxisrufsystem verwendet werden kann. Diese unterstützt das Senden und Empfangen vorkonfigurierter Benachrichtigungen. Die Applikation wurde auf Basis des Frameworks Nativescript als Multi-Platform Applikation entwickelt [2]. Dadurch ist es möglich, dieselbe Codebasis für die Entwicklung von Android und iOS Applikationen zu verwenden [5]. Im Fazit des Vorgängerprojekts wurde empfohlen, diese Applikation durch native Applikationen pro Platform zu ersetzen. Dadurch können effiziente Weiterentwicklung, sowie Hardware- und Betriebssystemkompatibilität langfristig gewährleistet werden [2].

Mit diesem Projekt wird die Applikation deshalb neu als native iOS Applikation umgesetzt. Dabei ist es wichtig, dass sämtliche bestehende Funktionalität auch im neu entwickelten nativen Mobile Client zur Verfügung steht. Um weiterhin Benachrichtigungen empfangen zu können, muss die gewählte Technologie es ermöglichen, Firebase Cloud Messaging anzubinden. Um Benachrichtigungen versenden und Konfigurationen laden zu können, muss es möglich sein, die Applikation an die API des Cloudservices anzubinden. Weiter muss die Technologie es ermöglichen, regelmäßige Aufgaben auszuführen. Dadurch kann überprüft werden, ob der Benutzer ungelesene Benachrichtigungen hat und wenn nötig eine Erinnerung dafür angezeigt werden.

#### 5.1.1 Programmiersprache

Für die Entwicklung von nativen iOS Applikationen ist die Programmiersprache Swift Industriestandard [6]. Der native iOS Client für Praxisruf wird deshalb mit Swift implementiert.

#### 5.1.2 Frameworks

Für die Umsetzung von iOS Applikationen stellt Apple die zwei Frameworks UIKit [7] und SwiftUI [8] zur Verfügung. UIKit ist das ältere der beiden Frameworks und ist seit iOS 2.0 verfügbar. Dementsprechend ist das Framework ausgereift und bietet viele Funktionen zur Integration einer Applikation mit iOS [7].

SwiftUI ist deutlich neuer und steht seit iOS 13.0 zur Verfügung [8]. Apple wirbt auf der offiziellen Dokumentationsseite für SwiftUI und schreibt: "SwiftUI helps you build great-looking apps across all Apple platforms with the power of Swift — and as little code as possible." [8]. SwiftUI fokussiert sich auf eine deklarative Syntax und ist dadurch leichtgewichtiger als UIKit. Es bietet zudem ausgezeichnete Integration in die XCode Entwicklungsumgebung und viele Standardkomponenten wie Listenansichten, Formfelder und andere UIKomponenten [8]. Dadurch wird es einfacher eine Benutzeroberfläche mit nativem Look und Feel einer iOS Applikation umzusetzen.

Da SwiftUI deutlich neuer ist als UIKit, ist es möglich das es noch nicht alle Funktionen und Betriebssystem Integrationen unterstützt die in UIKit möglich sind. Dieses Problem wird dadurch relativiert, dass UIKit Funktionen nahtlos in SwiftUI integriert werden können [9]. Es ist also grundsätzlich möglich alles, was mit UIKit umgesetzt werden kann, auch mit SwiftUI umzusetzen.

### 5.1.3 Anbindung Firebase Cloud Messaging

Die neue iOS Applikation, muss es weiterhin ermöglichen Benachrichtigungen über Firebase Cloud Messaging zu empfangen. Firebase stellt dazu eine native iOS Bibliothek zur Verfügung [10]. Die Integration von Firebase Cloud Messaging kann mit dieser Library implementiert werden. Dies beinhaltet die Registrierung bei Firebase Cloud Messaging, sowie das Empfangen der Benachrichtigungen über Firebase [11]. Damit Push-Benachrichtigungen über das Betriebssystem angezeigt werden können, müssen empfangene Benachrichtigungen an das Betriebssystem übergeben werden. Mit sogenannten AppDelegates ist es möglich, mit dem iOS-Betriebssystems zu interagieren. Dies beinhaltet das Anzeigen von Vorder- und Hintergrundbenachrichtigungen über das Benachrichtigungszenter von iOS anzusehen [12].

Die Firebase Cloud Messaging Bibliothek kann sowohl mit SwiftUI als auch mit UIKit verwendet werden. Die Integration muss dabei aber auf jeden Fall über einen AppDelegate umgesetzt werden [11]. AppDelegate sind ein Konzept welches aus UIKit stammt und in reinen SwiftUI Applikationen nicht verwendet werden [12]. UIKit Funktionen können allerdings nahtlos mit SwiftUI integriert werden. So ist es auch möglich AppDelegate in SwiftUI Anwendungen zu verwenden [9]. Die Firebase Cloud Messaging Bibliothek für iOS ermöglicht es damit, Benachrichtigungen von Praxisruf sowohl mit UIKit als auch mit SwiftUI umzusetzen.

### 5.1.4 Anbindung Cloudservice API

Die iOS Applikation muss die API des Cloudservice ansprechen können, um Konfigurationen zu laden und Benachrichtigungen zu versenden. Die bestehende API des Cloudservice ist als HTTP-Schnittstelle umgesetzt [2]. Die Integration von Http Anfragen in nativen iOS Applikationen ist über die URLRequest-Komponente aus der iOS Standardbibliothek möglich. Diese erlaubt das Erstellen von beliebigen Http-Requests [13]. Dies bedeutet, dass die Anbindung an die API des Cloudservice kann mit Mitteln der iOS Standardbibliothek umgesetzt werden.

### 5.1.5 Benachrichtigungen prüfen

Die mobile Applikation aus dem Vorgängerprojekt sammelt empfangene Benachrichtigungen in einer Inbox. Praxismitarbeitende müssen Benachrichtigungen in dieser Inbox als gelesen markieren. Die App prüft regelmäßig, ob ungelesene Benachrichtigungen vorhanden sind. Ist dies der Fall, wird ein Benachrichtigungston zur Erinnerung abgespielt. Diese Funktion muss auch von der neuen iOS Applikation unterstützt werden.

Die iOS Standardbibliothek bietet Mittel, mit welchen regelmässige Aufgaben angestossen werden können. Einerseits können über eine Timer-Komponente in regelmässigen Abständen Events veröffentlicht werden. Auf einer SwiftUI Ansicht (View) können Komponenten (Listener) registriert werden, die beim Empfang eines Timer-Events aufgerufen werden. Dies bringt die Einschränkung mit sich, dass die Prüfung nur ausgeführt ist, wenn die App im Vordergrund läuft und die View geladen wurde [14]. Da der bestehende Mobile Client dieselbe Einschränkung hat, kann mit einem Timer dasselbe Verhalten, wie in der bestehenden Applikation, umgesetzt werden. Um die Prüfung auch im Hintergrund auszuführen, kann die Komponente BGTaskScheduler verwendet werden. Über diese können Aufgaben erfasst werden, die im Hintergrund ausgeführt werden [15].

Die Erinnerungsfunktion kann mit Mitteln aus der iOS Standardbibliothek umgesetzt werden. Dabei ist es möglich die Prüfung analog zur bestehenden Applikation umzusetzen und auszuführen, solange die Applikation angezeigt wird. Weiter kann die Prüfung in Zukunft erweitert werden, um auch nach dem Beenden der Applikation an ungelesene Benachrichtigungen zu erinnern.

### 5.1.6 Entscheid

Die native iOS Applikation für Praxisruf wird mit Swift und SwiftUI umgesetzt. Für die Anbindung an Firebase Cloud Messaging wird die nativen iOS Bibliothek von Firebase umgesetzt. Die Anbindung an die API des Cloudservice und die Umsetzung der Erinnerungsfunktion werden mit Mitteln der iOS Standardbibliothek umgesetzt.

Die deklarative Syntax von SwiftUI ermöglicht es, einfacher übersichtliche und lesbare Komponenten zu implementieren. Integration in die XCode Entwicklungsumgebung und verfügbare Standardkomponenten vereinfachen diese Entwicklung weiter. Es ist möglich, dass einige Funktionen noch nicht mit SwiftUI umgesetzt werden können, weil dafür benötigte Funktionen noch nicht unterstützt werden. Da UIKit Funktionen nahtlos in SwiftUI integriert werden können, ist es möglich betroffene Teile der Applikation mit UIKit zu implementieren. Diese Teile können, sobald die entsprechenden Funktionen in SwiftUI verfügbar sind, migriert werden. Um die Verwendung von UIKit Funktionen auf ein Minimum zu reduzieren wird als Zielplattform iOS15 verwendet. Damit wird die zur Zeit der Umsetzung neueste iOS Version unterstützt. Dies ermöglicht es, alle verfügbaren SwiftUI Features zu verwenden und minimiert die Wahrscheinlichkeit, auf UIKit zurückgreifen zu müssen.

## 5.2 Sprachsynthese

In diesem Kapitel wird entschieden, mit welcher Technologie das Feature Sprachsynthese umgesetzt wird. Das Feature fordert, dass das System das Vorlesen empfangener Benachrichtigungen unterstützt. Um dies zu ermöglichen, muss eine Technologie integriert werden, die es erlaubt Sprachdaten aus Textinhalten zu synthetisieren. Diese Sprachdaten müssen als Audiodateien von der iOS Applikation abgespielt werden können.

Diese Anforderungen können mit den Standardbibliotheken für iOS-Entwicklung oder durch die Anbindung eines externen Anbieters umgesetzt werden. Die Anbindung eines externen Anbieters kann direkt im Mobile Client implementiert werden. Alternativ kann die Serverkomponente Cloudservice an einen Anbieter angebunden werden und allen Clients eine einheitliche Schnittstelle bieten, um diese Daten abzufragen.

### 5.2.1 Apple Speech Synthesis

Die iOS Standardbibliothek bietet Komponenten zur Konvertierung von Text zu Sprache [16]. Die Verwendung dieser Komponenten verspricht zwei Vorteile: Erstes kann Sprachsynthese dadurch ohne die Anbindung eines Drittanbieters umgesetzt werden. Zweitens ist die Kompatibilität mit iOS15 Clients garantiert und die Integration der Funktionen ohne externe Bibliotheken möglich [16]. Gleichzeitig entsteht damit aber eine starke Bindung an Apple als Dienstleister für Sprachsynthese. Sollte die Funktion in künftigen Versionen nicht mehr unterstützt werden, müsste die ganze Integration von Sprachsynthese neu evaluiert und implementiert werden. Weiter reduziert diese Variante die Flexibilität der Systemarchitektur. Mit der Verwendung der iOS Standardbibliothek für Sprachsynthese, muss die Anbindung an den Dienstleister direkt in der iOS Applikation umgesetzt werden. Dies ist insbesondere ein Nachteil, da für dieselbe Funktionalität in zukünftigen Android Clients ein anderer Dienstleister verwendet werden muss. Eine einheitliche Integration der Sprachsynthese in zukünftigen Plattformen ist deshalb nicht möglich, wenn Apple Speech Synthesis verwendet wird.

### 5.2.2 Amazon Polly

Die Anforderung U19<sup>1</sup> fordert, dass für Infrastruktur und Dienstleistungen wo möglich Amazon Webservices verwendet wird. Mit dem Amazon Polly bietet Amazon Webservices einen Dienst, welcher Text in Sprachdaten verwandeln kann [3]. Amazon Webservices stellt Libraries für iOS [17], Android [18] und für Java zur Verfügung [19]. Amazon Polly kann damit sowohl direkt in native mobile Applikationen als auch in den Cloudservice integriert werden.

Die iOS Library von Amazon Polly ermöglicht es, die Anbindung des externen Sprachsyntheseproviders direkt im Mobile Client zu implementieren [17]. Diese Lösung kann für zukünftige Android Clients analog mit der Android Library für Aws Polly umgesetzt werden. Die starke Bindung zu Apple als Dienstleister für Sprachsynthese entfällt durch diese Lösung. Es wird allerdings eine starke Bindung zu Amazon Polly als Dienstleister geschaffen. Ein Wechsel des Anbieters bleibt auch in dieser Variante aufwändig. Insbesondere, wenn in Zukunft Clients für weitere Plattformen angeboten werden.

Mit der Java Bibliothek von Amazon Polly kann die Anbindung des Sprachsyntheseproviders im Cloudservice vorgenommen werden. Dadurch ist es möglich im Cloudservice eine Schnittstelle zu implementieren, welche den Bezug von Sprachdaten für Benachrichtigungen erlaubt. Diese Schnittstelle kann in die bestehende API des Cloudservices integriert werden. Damit können alle Clients die Sprachdaten über dieselbe Schnittstelle beziehen. Caching von Sprachdaten kann sowohl auf Serverseite als auch auf Clientseite implementiert werden. Die Integration von Sprachsynthese in die API des Cloudservices ermöglicht es weiter, den gewählten Dienstleister in Zukunft einfach und ohne die Clients anzupassen

---

<sup>1</sup>Siehe Kapitel 3.1.3

auszutauschen. Gleichzeitig hat diese Variante den Nachteil, dass der Cloudservice komplexer wird. Durch die Integration von Sprachsynthese wird ein neues Umsystem angebunden. Die Komplexität des Systems als Ganzes, wächst dadurch in jedem Fall. Wie stark die Komplexität des Systems zunimmt, kann jedoch minimiert werden, indem die neue Funktionalität eng gekapselt wird. Sie kann so umgesetzt werden, dass sie unabhängig vom restlichen System bleibt und alle nötigen Daten über die Schnittstellen anderer Module bezieht.

Als dritte Option für die Integration von Amazon Polly kann AWS Lambda verwendet werden [3]. AWS Lambda erlaubt es Funktionalität ohne Serverinstanz auszuführen. Der Bezieher der Dienstleistung definiert die auszuführende Funktionalität und bestimmt, wann diese ausgeführt wird. Die Entscheidung, wie die Funktionalität ausgeführt wird, wird dabei an AWS übergeben [20]. Dies erlaubt es Funktionalität in ein System zu integrieren, ohne Serverinfrastruktur aufzubauen zu müssen.

Die Integration von Amazon Polly über AWS Lambda kann für Praxisruf in zwei Verarbeitungsschritten implementiert werden. In einem ersten Schritt werden die zu synthetisierenden Daten geladen und an Amazon Polly gesendet. Anschliessend werden die Resultate, die Amazon Polly liefert persistiert. Die Abfrage von Sprachdaten kann in diesem Fall ebenfalls in die API des Cloudservices integriert werden. Dazu muss der Cloudservice auf die persistierten Sprachdaten zugreifen und einen Endpunkt anbieten, um diese abzufragen. Dieser Ansatz bringt zwei Nachteile mit sich. Erstens müssen die synthetisierten Sprachdaten zwingend ausserhalb des Mobile Clients persistiert werden. Zweitens wird eine stärkere Bindung an Amazon als Dienstleister geschaffen, weil die Anbindung der Sprachsynthese über proprietäre Funktionalität von Amazon implementiert wird. Gleichzeitig bietet der Ansatz den Vorteil, dass weniger Infrastruktur benötigt wird, da die Abfrage an AWS Polly ohne dedizierten Server abgesetzt werden kann. Um eine einheitliche API zu bieten, muss das System aber auch in diesem Fall erweitert werden, um eine entsprechende Schnittstelle anbieten zu können.

### 5.2.3 Entscheidung

Die Sprachsynthese wird durch die Anbindung des externen Anbieters Amazon Polly umgesetzt. Der Cloudservice übernimmt die Kommunikation mit Amazon Polly und bietet eine Schnittstelle, über welche Clients Audiodaten beziehen können. Diese Schnittstelle wird in die API des Cloudservices integriert. Die Anbindung an Amazon Polly wird dabei direkt im Cloudservice implementiert und nicht über AWS Lambda gelöst. So kann die bestehende Infrastruktur des Cloudservices übernommen werden und es ist nicht notwendig die Sprachdaten ausserhalb von Mobile Clients zu persistieren. Dies erlaubt es einerseits ein Clientseitiges Cache für Sprachdaten zu implementieren und dadurch Netzwerkanfragen zu minimieren. Es erlaubt es weiter, nur Sprachdaten zu synthetisieren, welche tatsächlich von Clients angefragt werden.

Durch diesen Ansatz wird die Abhängigkeit zu einzelnen Anbieter minimiert. Die Anbindung von Sprachsynthese kann für alle Client-Plattformen einheitlich umgesetzt werden. Dies macht die gewählte Variante zukunftssicher und bietet grosse Flexibilität für den Betrieb. Der Zuwachs an Komplexität durch die Anbindung eines neuen Umsystems wird durch entsprechende Kapselung der Funktionalität in ein eigenes Modul minimiert.

### 5.3 Gegensprechanlage

Die zentrale Aufgabenstellung dieser Arbeit dreht sich darum, Peer-To-Peer Sprachübertragung in ein cloudbasiertes Praxisrufsystem zu integrieren. In diesem Kapitel wird entschieden, welche Technologie zur Umsetzung dieser Aufgabe verwendet wird. Es werden dabei zwei Ansätze beschrieben, mit welchen diese Anforderung umgesetzt werden kann. Der erste Ansatz beinhaltet die Anbindung eines externen Anbieters für Kommunikationslösungen. Der zweite Ansatz verzichtet auf die Integration eines Anbieters und setzt Sprachverbindungen als Peer-To-Peer Verbindungen basierend auf WebRTC um.

#### 5.3.1 Amazon Chime

Eine Möglichkeit um Sprachverbindungen in Praxisruf zu implementieren, ist die Integration einer bestehenden Business-Kommunikationslösung wie Webex oder Microsoft Teams. Auch für die Integration von Sprachverbindungen gilt, dass für Infrastruktur und Dienstleistungen wo möglich Amazon Webservices verwendet wird<sup>2</sup>. Mit Amazon Chime bietet Amazon einen Dienst für Telefonie, Chats und Videokonferenzen [21]. Chime bietet einerseits Web- und Mobile Applikationen für Anrufe und Meetings. Andererseits ist es mit dem Amazon Chime SDK für iOS möglich, Chime in eigene native iOS Applikationen zu integrieren [22]. Chime ermöglicht es Konferenzunterhaltungen mit bis zu 250 Teilnehmern zu starten [23]. Damit bietet der Anbieter alle benötigte Funktionen um Einzel- und Gruppenunterhaltungen in einem Praxisrufsystem umzusetzen.

Integration eines Providers wie Amazon Chime hat den Vorteil, dass die Telefonie in einen etablierten Provider ausgelagert werden kann. Durch Verträge mit dem Provider können Verfügbarkeitsgarantien und Supportleistungen vereinbart werden. Dies erhöht die Stabilität des Systems und ermöglicht effizientes Reagieren im Fehlerfall. Weiter fallen für Praxisruf selbst wenig bis keine Aufwände für den Betrieb der nötigen Infrastruktur.

Amazon Chime unterstützt deutlich mehr Funktionen als in einem Praxisrufsystem benötigt werden. Ein Rufsystem muss als Gegensprechanlage verwendet werden können und kurze Gespräche zwischen Teilnehmern erlauben. Dies bedeutet einerseits, dass Chime garantiert alle Funktionen bietet welche benötigt werden. Es bedeutet aber andererseits auch, dass nur ein kleines Subset der vorhandenen Möglichkeiten ausgenutzt wird. Ein Nachteil daran ist, dass eine starke Bindung an Amazon als Telefonieanbieter und die Infrastruktur, die Amazon Chime zum Verbindungsaufbau anbietet, stattfindet. Gleichzeitig kann von vielen Vorteilen von Amazon Chime nicht profitiert werden, weil die entsprechenden Funktionen für ein Praxisrufsystem nicht relevant sind.

#### 5.3.2 WebRTC

WebRTC (Web Real-Time Communication) ermöglicht Echtzeitkommunikation basierend auf einem offenen Standard. Das Open-Source-Projekt wird unter anderem von Apple, Google, Microsoft und Mozilla unterstützt. Es erlaubt den Austausch von Sprach-, Video- und allgemeinen Daten zwischen Clients. Wie auf der offiziellen Webseite von WebRTC beschrieben stehen die Technologien "[...] in allen gängigen Browsern als reguläre JavaScript-APIs verfügbar. Für native Clients wie Android- und iOS-Anwendungen steht eine Bibliothek zur Verfügung, die dieselben Funktionen bietet." [4].

WebRTC baut zur Kommunikation direkte Peer-To Peer-Verbindungen zwischen den Kommunikationspartnern auf. Die WebRTC Libraries und APIs bieten Komponenten für die Integration von Peer-To-Peer Verbindung, Hardwarezugriff auf Microphon und Kamera. WebRTC spezifiziert allerdings nicht, wie für den Verbindungsaufbau notwendige Informationen zwischen Teilnehmern ausgetauscht werden müssen.

---

<sup>2</sup>Siehe Anforderung U19 in Kapitel 3.1.3

Um den Austausch dieser Informationen zu ermöglichen ist eine Signaling Instanz notwendig. Die Signaling Instanz muss es ermöglichen, Informationen zwischen Teilnehmern zu vermitteln. WebRTC spezifiziert nicht, wie diese Signaling Instanz aussehen muss. Neben der Signaling Instanz ist keine weitere Infrastruktur notwendig. Sämtliche Daten werden direkt über die Peer-To-Peer Verbindungen zwischen Clients ausgetauscht.

Dies hat den Vorteil, dass die Abhängigkeit zu externen Umsystemen minimiert werden kann. Die Signaling Instanz kann auf die eigenen Anforderungen zugeschnitten werden. Die Technologie über welche Signale ausgetauscht werden, kann frei gewählt werden. Signaling für Sprachverbindungen im Praxisrufsystem kann damit in den Cloudservice integriert werden. So kann notwendige Infrastruktur, die als Teil des Praxisrufsystems betrieben werden muss, schlank gehalten werden. Die Integration der Signalvermittlung in den Cloudservice bietet weiter den Vorteil, dass bestehende Funktionen des Systems angesprochen werden können. So können z.B. Benachrichtigung bei verpassten Anrufen über die bereits umgesetzte Benachrichtigungsfunktion versendet werden.

Die Verwendung von WebRTC hat den Nachteil, dass kein fachlicher Support bei Verbindungsproblemen zur Verfügung steht. Signaling Instanz und Implementation der Sprachverbindungen stehen in der alleinigen Verantwortung des Betreibers des Praxisrufsystems. Abgesehen von Fehlern in der Implementation können Verbindungsprobleme an zwei Stellen auftreten. Einerseits ist es möglich, dass die Signaling Instanz ausfällt und nicht erreichbar ist. Dieses Problem kann durch den Betrieb des Cloudservice adressiert werden. Die Signaling Instanz kann in den Cloudservice integriert werden. Dieser wird wiederum bei einem Cloud-Provider betrieben. Für diesen Betrieb können Verträge definiert werden, die Verfügbarkeit und Support im Fehlerfall bieten. Andererseits können lokale Netzwerkprobleme auftreten, welche Endgeräte unerreichbar machen. Dieses Problem besteht unabhängig davon, wie Sprachübertragung implementiert wird. Im Fall von Praxisruf kann das Problem dadurch adressiert werden, dass nicht erreichbare Endgeräte durch Benachrichtigungen über verpasste Verbindungen informiert werden.

Die Gegensprechanlage in einem Praxisrufsystem muss Verbindung mit mehreren Teilnehmern gleichzeitig erlauben. Verbindungen müssen deshalb als zwischen zwei Teilnehmern als 1:1 Verbindungen, aber auch zwischen mehreren Teilnehmern als 1:n Verbindungen umgesetzt werden. WebRTC sieht ausschließlich direkte Peer-To-Peer Verbindungen vor. Es ist allerdings möglich, mehrere direkte Peer-To-Peer Verbindungen gleichzeitig zu verwenden [24]. Weiter ist es möglich, die Kommunikation über eine zentrale Instanz zu bündeln. Eine solche nennt sich Multipoint Conferencing Unit (MCU). Bei der Verwendung einer MCU werden Verbindungen nicht direkt zwischen Endgeräten hergestellt. Stattdessen stellt jedes Endgerät eine Verbindung mit der MCU her. Die MCU vermittelt die Daten zwischen allen Teilnehmern. Die Verwendung einer MCU verkompliziert das System und die benötigte Infrastruktur massgeblich [25].

### 5.3.3 Entscheidung

Sprachverbindungen für die Funktion Gegensprechanlage werden mit WebRTC umgesetzt. Der Cloudservice wird um ein Modul erweitert, welches als Signaling Instanz dient. Die WebRTC iOS Bibliothek wird verwendet, um Peer-To-Peer Sprachübertragung in einer nativen iOS App zu implementieren.

Durch die Verwendung von WebRTC, können Sprachverbindungen aufgebaut werden, ohne ein weiteres Drittsystem anzubinden. Die benötigte Infrastruktur kann dadurch auf ein Minimum reduziert werden. Eine eigene Implementierung der Signaling Instanz ermöglicht es weiter, bestehende Funktionen des Praxisrufsystems effizient zu verwenden. Die Verfügbaren Libraries für iOS, Android und Javascript bedeuten, dass künftige Web- und Android Clients in das System integriert werden können. Die Verfügbarkeit der Signaling Instanz wird den Betrieb des Cloudservices sichergestellt.

## 6 Peer-To-Peer Sprachübertragung

Dieses Kapitel beschreibt die Funktionsweise von WebRTC und bildet die theoretische Grundlage dafür, wie Peer-To-Peer Sprachübertragung mit WebRTC in einem Praxisrufsystem eingesetzt werden kann.

### 6.1 Verbindungsprotokolle

Im Folgenden werden die zentralen Kommunikationsprotokolle beschrieben, welche von WebRTC verwendet werden. Die wichtigsten Protokolle im WebRTC Umfeld sind das Session Description Protocol und das Interactive Connectivity Establishment.

Das Session Description Protocol (SDP) dient als Beschreibung einer Verbindung über welche Multimediadaten wie Audio oder Video übertragen werden [26]. Diese Beschreibung beinhaltet Metainformationen zu einer Verbindung. Dazu gehören unter anderem Auflösung, Format sowie Verschlüsselung und Codierung der zu übertragenden Daten [27]. WebRTC verwendet SDP, um zu beschreiben welche Art von Daten mit einer Verbindung übertragen werden und wie diese verarbeitet werden können [28]. Um eine Verbindung mit WebRTC aufzubauen, müssen die Teilnehmer der Verbindung SDP Informationen austauschen. Dieser Austausch bildet den ersten Teil an Signalmeldungen, welche über die Signaling Instanz ausgetauscht werden müssen.

Das Protokoll Interactive Connectivity Establishment (ICE) ermöglicht es, Netzwerkverbindungen zwischen Endgeräten aufzubauen. Es wird verwendet, um eine möglichst direkte Netzwerkverbindung zwischen zwei Teilnehmern herzustellen. Zu diesem Zweck tauschen die Teilnehmenden sogenannte ICE Candidates aus. Ein ICE Candidate beschreibt eine mögliche Verbindung, über welche die Kommunikation stattfinden kann [29]. ICE wird in WebRTC verwendet, um die Netzwerkverbindung zwischen zwei Teilnehmern aufzubauen [27]. Der Austausch von ICE Candidates bildet den zweiten Teil an Signalmeldungen, welche über die Signaling Instanz ausgetauscht werden müssen [28].

Das ICE-Protokoll kann die Protokolle "Session Traversal Utilities for NAT" (STUN) und "Traversal Using Relays around NAT" (TURN) verwenden. Beide Protokolle erlauben es in IP-Netzwerken hinter Firewalls und Network Address Translators (NAT) Peer-To-Peer Verbindungen aufzubauen [29]. NAT wird verwendet, wenn die IP-Adresse eines Endgerätes im lokalen Netzwerk nicht von außerhalb dieses Netzwerks erreichbar ist. NAT ermöglicht es, solchen Endgeräten mit Zielen außerhalb des privaten Netzwerkes zu kommunizieren [30]. Das Protokoll STUN erlaubt den Umgang mit NAT, indem es Clients erlaubt, die eigene öffentliche IP-Adresse zu ermitteln. Um dies zu ermöglichen wird ein STUN-Server außerhalb des lokalen Netzwerkes angefragt. Aufgrund der Adresse, welche dieser Server zurück liefert, kann ein Ice Candidate erstellt werden [29]. Mit dem Protokoll TURN ist es möglich Verbindungen aufzubauen, wenn weitere Restriktionen vorhanden sind, die Netzwerkverbindungen einschränken. Dabei dient ein TURN-Server als vermittelnde Instanz, welche auszutauschenden Daten zwischen den Endgeräten weiterleitet [31].

Die Protokolle STUN und TURN sind nur notwendig, wenn keine direkte Verbindung zwischen den beteiligten Endgeräten möglich sind. Wenn alle beteiligten Endgeräte im selben lokalen Netzwerk sind und direkte Verbindungen aufbauen können, kann ein ICE Candidate mit der lokalen Adresse der Beteiligten erstellt werden [31].

### 6.2 Verbindungsaufbau

Die Protokolle SDP und ICE werden in Endgeräten verwendet, um Verbindungen aufzubauen. Als zentrale Schnittstelle für diese Protokolle im Endgerät definiert der Standard das Interface RTCPeerConnection. Diese Schnittstelle repräsentiert die Peer-To-Peer Verbindung zu einem anderen Endgerät. Sie muss sowohl auf dem Gerät, welches einen Anruf startet, als auch auf dem Zielgerät initialisiert werden [28].

Die WebRTC Bibliothek für iOS implementiert dieses Interface. Es bietet die Möglichkeit, eine RTC-PeerConnection für die gewünschte Übertragung zu initialisieren. Nach dieser Initialisierung ist noch keine Verbindung zu einem anderen Endgerät aufgebaut. Es ist ausschliesslich ein lokales Verbindungsobjekt initialisiert. Aus diesem Verbindungsobjekt können die nötigen SDP Informationen generiert werden. Diese Informationen müssen über die Signaling Instanz an das Zielgerät übertragen werden. Die Signalmeldung, welche SDP Informationen des Initiators der Verbindung überträgt, stellt ein Offer-Signal dar. Dieses kann darauf die RTCPeerConnection auf seiner Seite initialisieren und entsprechende SDP Informationen über die Signaling Instanz zurücksenden. Die Signalmeldung, welche die SDP Informationen des Verbindungsziels an den Sender übermittelt, stellt ein Answer-Signal dar.

Damit die Peer-To-Peer Verbindung aufgebaut werden kann, müssen die ICE Candidates gemäss dem ICE Protokoll ermittelt werden. Wenn eine direkte Verbindung zwischen beiden beteiligten Geräten aufgebaut werden kann, sind keine weiteren Anfragen notwendig. Ein ICE Candidate für die direkte Verbindung kann erstellt und über die Signaling Instanz geteilt werden. Ist keine direkte Verbindung möglich, müssen mögliche ICE Candidates von einem ICE Server ermittelt werden. Dazu können die Protokolle STUN oder TURN eingesetzt werden. Die Abfragen an den ICE Server werden von der Implementation der RTCPeerConnection übernommen. Verfügbare ICE Server und das für die Abfrage zu verwendende Protokoll müssen bei Initialisierung der RTCPeerConnection mitgegeben werden. ICE Candidates werden von beiden Teilnehmern erstellt und über die Signaling Instanz geteilt. Dieser Prozess wird wiederholt, bis sich beide Teilnehmer auf einen ICE Candidate geeinigt haben.

### 6.3 Signaling

Der Aufbau von Peer-To-Peer Sprachverbindungen mit WebRTC erfordert das Austauschen von Signalmeldungen. Dabei sind wie in Kapitel 6.2 beschrieben mindestens die drei Signale Offer, Answer und Ice Candidate notwendig. Um diesen Austausch zu ermöglichen ist eine zentrale Signaling Instanz notwendig. Diese muss Signale entgegennehmen und an relevante Empfänger weiterleiten können [28].

Der WebRTC Standard schreibt nicht, vor wie eine Signaling Instanz umgesetzt werden muss. Er definiert einzig, welchen Inhalt Signale für den Verbindungsaufbau beinhalten müssen und wie dieser Inhalt verarbeitet werden muss [28]. Konzept und Funktionsweise der Signaling Instanz und Signalmeldungen in Praxisruf werden in Kapitel 7 beschrieben.

### 6.4 Sicherheit

Bei Sprachverbindungen mit WebRTC gibt es zwei sicherheitskritische Kanäle. Erstens muss der Kanal, über welchen Signalmeldungen ausgetauscht werden gesichert sein. Nur Berechtigte dürfen Signalmeldungen versenden und empfangen. Dazu muss die Identität der Teilnehmer, die Signalmeldungen austauschen verifiziert werden. Aussenstehende und nicht Berechtigte dürfen den Inhalt von Signalmeldungen nicht lesen können [32]. Die Implementierung dieser Mechanismen muss vom Anbieter der Signaling Instanz umgesetzt werden [28].

Zweitens muss die Verbindung über welche Echtzeitdaten ausgetauscht werden gesichert sein. Diese Datenverbindungen werden über das Protokoll Datagram Transport Layer Security (DTLS) verschlüsselt. Bevor Daten über eine Verbindung ausgetauscht werden, wird pro Übertragungskanal ein DTLS-Handshake ausgeführt. Über diesen Handshake werden die nötigen Schlüsselinformationen ausgetauscht. Diese Schlüssel werden anschliessend für sichere Datenübertragung über das Protokoll "Secure Real-Time Transport" verwendet [32]. Die Implementierung dieser Mechanismen muss von der Bibliothek, welcher den Standard implementiert umgesetzt werden [28].

## 6.5 Unicast, Multi-Cast, Broadcast

Dieses Kapitel beschreibt, wie der Verbindungsaufbau von Peer-To-Peer Verbindungen mit WebRTC als Uni-, Multi- und Broadcast in einem cloudbasierten Praxisrufsystem umgesetzt werden kann. Dabei wird davon ausgegangen, dass wie in Kapitel 4 entschieden, keine Multipoint Conferencing Unit eingesetzt wird.

WebRTC unterstützt ausschliesslich Peer-To-Peer Verbindungen. Einzelne Verbindungen können mit WebRTC immer nur zwischen genau zwei Teilnehmern bestehen. Gleichzeitig haben Signalmeldungen immer nur genau einen Empfänger. Aus der Sicht von WebRTC sind damit ausschliesslich Unicast Verbindungen möglich.

Um Multi- und Broadcast Signale zu erlauben, muss auf Applikationsstufe eingegriffen werden. Es ist mit WebRTC nicht möglich, eine Verbindung zwischen mehr als zwei Teilnehmern aufzubauen. Es ist hingegen möglich, mehrere Verbindungen gleichzeitig aufzubauen und mit allen Teilnehmern gleichzeitig zu kommunizieren [24]. Dies bedeutet, dass der Verbindungsaufbau wie in Kapitel 6.2 beschrieben für jedes Ziel einzeln ausgeführt werden muss. Sowohl für Multi- als auch für Broadcast Verbindungen müssen die relevanten Ziele vor dem Verbindungsaufbau identifiziert werden. Im Fall eines cloudbasierten Praxisrufsystems kann dies über die zentrale Konfigurationsverwaltung gelöst werden.

Verbindung zu mehreren und/oder allen Teilnehmern können als 1:n Verbindungen umgesetzt werden. Dazu müssen die relevanten Teilnehmer durch die Applikation vor Verbindungsaufbau identifiziert werden. Anschliessend wird der Verbindungsaufbau auf der Seite des Initiators für jeden Empfänger wiederholt. Aus Sicht des Empfängers handelt es sich dabei um eine Unicast Verbindung. Für Empfänger gibt es damit keinen Unterschied zwischen 1:1 und 1:n Verbindungen. Bei einer solchen Verbindung kann der Initiator der Verbindung mit allen Empfängern kommunizieren. Jeder Empfänger kann allerdings nur mit dem Initiator kommunizieren. Eine Kommunikation zwischen einzelnen Empfängern ist in dieser Form nicht möglich.

Kommunikation zwischen allen Verbindungsteilnehmern wird möglich, wenn n:n Verbindungen verwendet werden. Dabei baut jeder Teilnehmer eine Verbindung zu jedem anderen Teilnehmer auf [24]. Im Rahmen dieser Arbeit sind lediglich 1:n Verbindungen gefordert. Konzepte und Umsetzung für n:n Verbindungen werden hier deshalb nicht weiter behandelt.

## 7 Konzept

Dieses Kapitel beschreibt das Konzept, nach welchem das cloudbasierte Praxisrufsystem aus dem Vorgängerprojekt erweitert wurde. Die Konzepte wurden vor Beginn der Umsetzung definiert und während der Umsetzung laufend überarbeitet und erweitert. Die folgende Dokumentation beschreiben die neuste Version des Konzepts und stellt damit das umgesetzte System dar. Dabei wird nicht hervorgehoben, welche Teile zu Beginn definiert und welche Teile später überarbeitet oder ergänzt wurden.

Das Konzept gibt zuerst einen Überblick über das System als Ganzes. Es werden die einzelnen Teile des Systems beschrieben und es werden Schritte definiert, um die Weiterentwicklung des Systems zu vereinfachen. Anschliessend werden Konzepte für die Umsetzung der drei Features "Native iOS Applikation", "Sprachsynthese" und "Gegensprechanlage" beschrieben.

### 7.1 Systemarchitektur

Dieses Kapitel beschreibt die Komponenten des cloudbasierten Praxisrufsystems und wie diese erweitert werden. Abbildung 7.1 gibt einen Überblick über die Systemkomponenten und welche Module diese beinhalten. Dabei sind Module, die im Rahmen dieses Projektes entwickelt oder neu integriert werden, grün umrandet. Die Pfeile zwischen Modulen zeigen gerichtete Abfragen und Aufrufe zwischen den Komponenten. Alle Abfragen zwischen Services sowie zwischen Modulen des Cloudservices finden über die API Schnittstellen der jeweiligen Module statt.

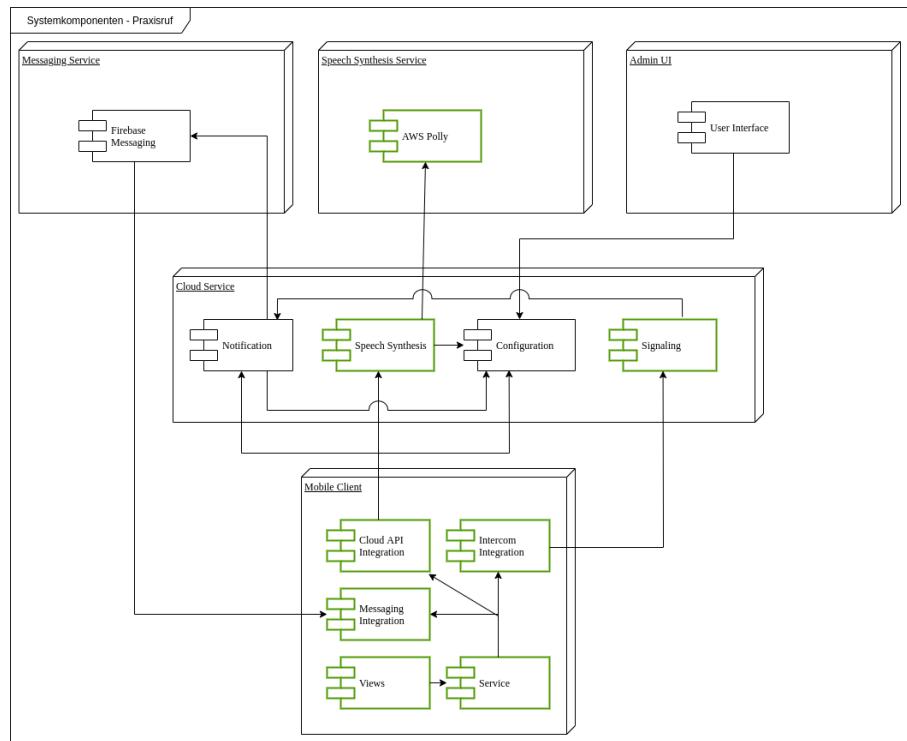


Abbildung 7.1: Systemkomponenten

#### 7.1.1 Systemkomponenten

Dieses Kapitel beschreibt die in Abbildung 7.1 abgebildeten Systemkomponenten. Dabei wird für jede Komponente beschrieben, welche Aufgaben ihr zufallen und wie sie im Rahmen dieses Projektes erweitert wird.

## Cloudservice

Der Cloudservice bildet die zentrale Serverkomponente von Praxisruf. Zu Beginn dieses Projektes umfasst der Cloudservice die beiden Domänen Notification und Configuration. Dabei ist die Domäne Notification für das Versenden von Benachrichtigungen und die Domäne Configuration für die Verwaltung und Auswertung der Konfigurationen verantwortlich. Die relevanten Empfänger für eine Benachrichtigung werden in der Configuration Domain ermittelt. Um auf diese Informationen zugreifen zu können, muss aus der Domäne Notification eine Abfrage an die Domäne Configuration gesendet werden. Die Configuration-Domäne bietet dazu eine HTTP Schnittstelle [2].

Die Trennung der Domänen wurde im Vorgängerprojekt lediglich auf Package-Ebene realisiert. Mit diesem Projekt soll die Trennung einen Schritt weiter gehen. Der Cloudservice wird in Module aufgeteilt. Diese Module werden weiterhin in einer einzelnen Applikation zusammengefasst, können aber zu einem späteren Zeitpunkt in einzelne Microservices betrieben werden.

Nachdem die Auftrennung der bestehenden Domänen in Module stattgefunden hat, wird der Cloudservice erweitert. Es werden die zwei Module Signaling und Speech Synthesis umgesetzt. Das neue Modul Signaling ist für die Signalvermittlung zwischen Mobile Clients verantwortlich. Es übernimmt die Aufgabe der Signaling Instanz für den Aufbau von Sprachverbindungen mit WebRTC. Das Modul Signaling hat eine funktionale Abhängigkeit zum Modul Notification. Es verwendet die API des Notification Moduls, um Empfänger mit Benachrichtigungen über verpasste Verbindungen zu informieren.

Das neue Modul Speech Synthesis dient als Schnittstelle zu einem externen Provider für Sprachsynthese. Dies ermöglicht es, die Sprachsynthese als Teil der API des Cloudservice anzubieten. Dadurch können Clients aller Plattformen und auch Systeme, die künftig angebunden werden, auf die Sprachsynthese zugreifen. Weil alle Clients die Daten aus derselben Schnittstelle beziehen, ist garantiert, dass die Konfiguration und Funktionsweise dieselbe für alle Clients ist.

## Mobile Client

Der Mobile Client ist eine mobile Applikation, über welche Praxisruf bedient werden kann. Der mit dem Vorgängerprojekt umgesetzte Mobile Client erlaubt es Benachrichtigungen zu versenden und empfangen [2]. Dieser Mobile Client wird durch eine native iOS Applikation ersetzt. Dabei wird sämtliche Funktionen des Mobile Clients aus dem Vorgängerprojekt in die native App migriert. Weiter werden die Funktionen Gegensprechanlage und Sprachsynthese für empfangene Benachrichtigungen umgesetzt.

## Admin UI

Das Admin UI ist eine Webapplikation, über welche die Konfiguration des Praxisrufsystems verwaltet werden kann. Die Konfiguration des Systems wird für Gegensprechanlage und Sprachsynthese erweitert. Für die Gegensprechanlage müssen Buttons konfiguriert werden können. Diese beinhalten Anzeigetext und Teilnehmer einer Sprachverbindung. Weiter muss pro Benachrichtigung konfigurierbar sein, ob ihr Inhalt bei Empfang vorgelesen werden sollen. Das Admin UI muss erweitert werden, um die Verwaltung der erweiterten Konfiguration zu ermöglichen.

## Messaging Service

Der Messaging Service ist für die Zustellung von Push Benachrichtigungen an Mobile Clients verantwortlich. Der Cloudservice muss an den Messaging Service angebunden sein, um Benachrichtigungen anhand der Konfiguration zu versenden [2]. Die Anbindung des Cloudservices an den Messaging Service ist mit dem Vorgängerprojekt bereits umgesetzt und muss für dieses Projekt nicht angepasst werden. Die neu entwickelte native iOS App muss hingegen an den Messaging Service angebunden werden, um Benachrichtigungen zu empfangen. Als Messaging Service wird Firebase Cloud Messaging verwendet.

## Speech Synthesis Service

Um Sprachsynthese zu ermöglichen, wird ein externer Service angebunden. Dieser übernimmt die Konvertierung von Text aus Benachrichtigungen zu Sprachdaten. Die Anbindung an den Speech Synthesis Service wird ausschliesslich im Cloudservice implementiert. Sämtliche andere Komponenten die Sprachdaten benötigen, fragen diese beim Cloudservice ab. Die REST API des Cloudservice wird um entsprechende Endpunkte erweitert. Als Speech Synthesis Service wird Amazon Polly verwendet.

### 7.1.2 Modularisierung Cloudservice

Der mit dem Vorgängerprojekt umgesetzte Cloudservice ist als monolithische Applikation implementiert. Er trennt intern die beiden Domänen Notification und Configuration. Die Domäne Configuration ist für die Verwaltung und Auswertung der Konfiguration des Systems und die Domäne Notification für das Versenden von Benachrichtigungen verantwortlich. Abhängigkeiten zwischen den beiden Domänen ist über eine REST-Schnittstelle abstrahiert.

Die Trennung der Domänen, erlaubt es die Anwendung zukünftig in mehrere Microservices aufzuteilen. Diese könnten unabhängig betrieben und erweitert werden. Weiter wird es dadurch möglich, einzelnen Teilen der Applikation mehr Ressourcen zuzuteilen. Die Trennung der Domänen in eigene Microservices wurde im Rahmen des Vorgängerprojektes noch nicht vorgenommen. Die beiden Domänen wurden lediglich durch die Package Struktur innerhalb eines einzelnen monolithischen Services getrennt.

Mit diesem Projekt wird die Trennung der Domänen innerhalb des Cloudservice verstärkt. Die Applikation wird in strikt getrennte Module aufgeteilt. Dabei wird pro Domäne ein Modul erstellt. Dieses kapselt sämtliche Domänenobjekte, Services und Schnittstellen der jeweiligen Domäne. Dadurch ist garantiert, dass die Domänen sauber voneinander getrennt sind. Sämtliche Kommunikation zwischen den Modulen muss über die API der jeweiligen Module geschehen.

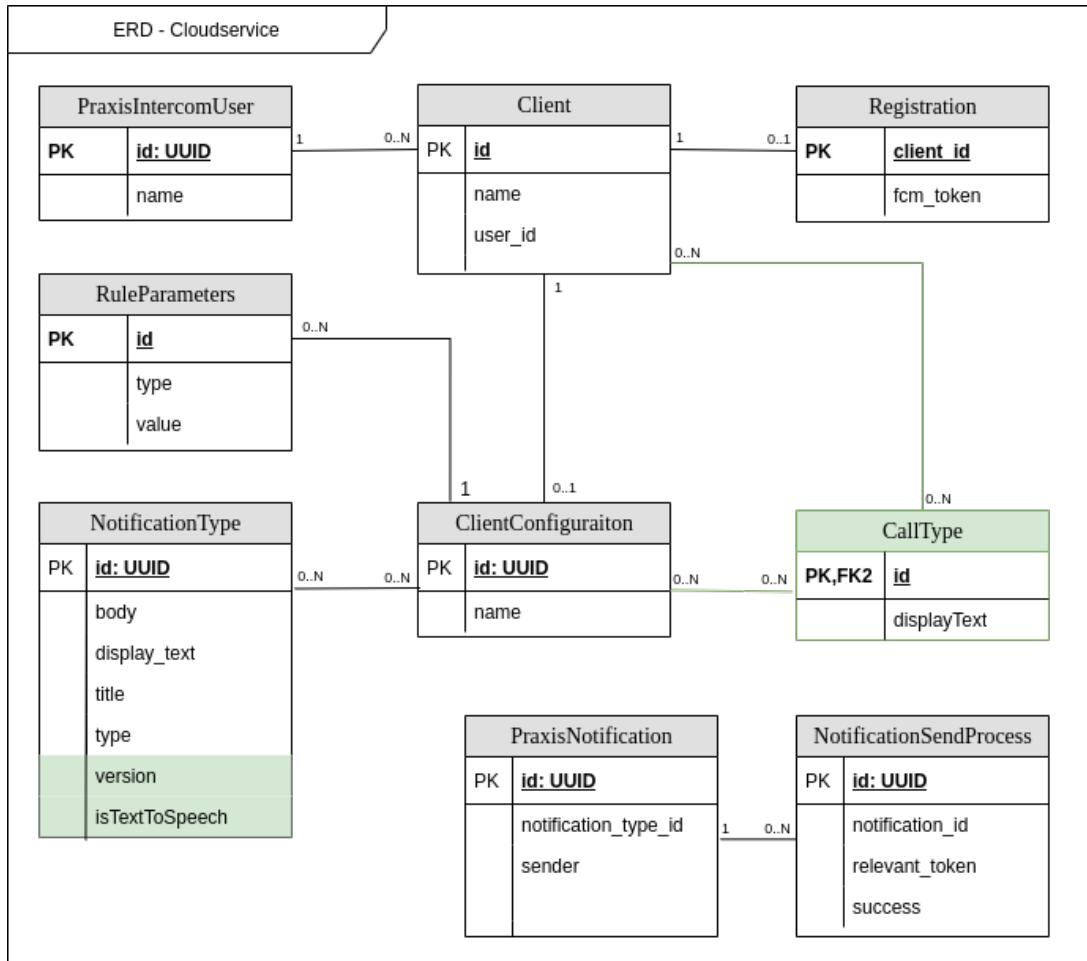
Es werden die vier Domänen-Module Configuration, Notification, Speech Synthesis und Signaling definiert. Das Modul Configuration beinhaltet alle Domänenobjekte, Services und Schnittstellen für die Verwaltung, Auswertung und Abfrage der Systemkonfiguration. Das Modul Notification beinhaltet alle Domänenobjekte, Services und Schnittstellen für das Versenden von Benachrichtigungen. Das Modul Speech Synthesis beinhaltet die Anbindung an den Speech Synthesis Service. Es stellt eine Schnittstelle zur Verfügung, über die Sprachdaten angefragt werden können. Das Modul Signaling dient als Signaling Instanz für den Aufbau von Sprachverbindungen mit WebRTC. Es beinhaltet Domänenobjekte, Services und Schnittstellen für die Signalvermittlung zwischen Mobile Clients.

Weiter werden die zwei Module Commons und App definiert. Komponenten, welche in mehr als einer Domäne verwendet werden, werden in ein zusätzliches Commons Modul verlegt. Dazu gehören Data Transfer Objects für Schnittstellen zwischen den Modulen, geteilte Clients um Abfragen auf andere Module abzusetzen sowie Komponenten Fehlerhandlung. Das Modul App hat eine Abhängigkeit auf alle Domänen Module und bietet ein Modul, über welches diese als einzelne Applikation gestartet werden können. Das Modul beinhaltet weiter die globale Security Konfiguration des Systems. Dadurch kann sichergestellt werden, dass diese für alle Systemteile angewendet wird.

Der Cloudservice wird weiterhin als monolithische Applikation betrieben. Die Modularisierung garantiert dabei eine strikte Trennung der Domänen. In Zukunft können einzelne Module aus dem Cloudservice ausgelöst und als eigenständige Applikationen betrieben werden.

### 7.1.3 Domänenmodell Cloudservice

Das Domänenmodell Cloudservices wird für die Integration von Sprachsynthese und Gegensprechkanal erweitert. Abbildung 7.2 zeigt das vollständige Domänenmodell der Domänen Configuration und Notification. Dabei sind Felder und Entitäten die in diesem Projekt hinzugefügt werden grün markiert. Die neuen Domänen Speech Synthesis und Signaling führen keine persistenten Daten. Die Services und Komponenten dieser Domänen sind in den Kapiteln 7.3 und 7.4 beschrieben.



**Abbildung 7.2:** Entitiy Relationship Diagramm - Cloudservice

## 7.2 Native iOS Applikation

Dieses Kapitel beschreibt das Konzept für einen nativen iOS Client zur Bedienung von Praxisruf. Es wird Entwurf und Funktionsweise der notwendigen Ansichten der Benutzeroberfläche beschrieben. Weiter wird definiert, wie die aus dem Vorgängerprojekt zu migrierenden Funktionen in eine native iOS Applikation integriert werden können. Dies beinhaltet insbesondere Anbindung der API des Cloudservice und Firebase Cloud Messaging. Die Integration von Gegensprechanlage und Sprachsynthese wird nur im Rahmen des Entwurfs der Benutzeroberfläche beschrieben. Eine detaillierte Beschreibung der Konzepte für diese Features folgt in den Kapiteln 7.3 und 7.4.

### 7.2.1 Benutzeroberfläche

Die Ansichten zur Anmeldung und Zimmerauswahl werden analog zum bestehenden Mobile Client umgesetzt. Die Loginseite (Abbildung 7.3) beinhaltet einen kurzen Willkommenstext und ein Logo für Praxisruf. Darunter findet sich ein einfaches Formular zur Eingabe von Benutzername und Passwort, sowie ein Button zur Bestätigung.



Abbildung 7.3: Mockup Login

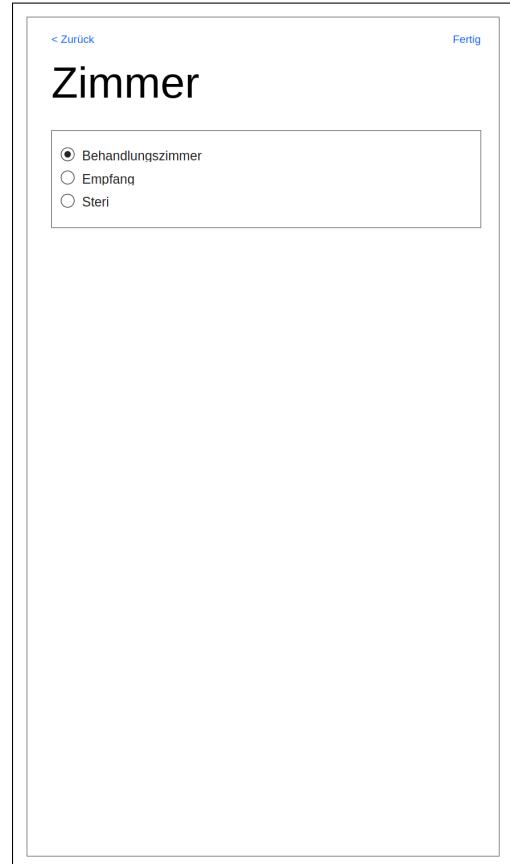


Abbildung 7.4: Mockup Zimmerwahl

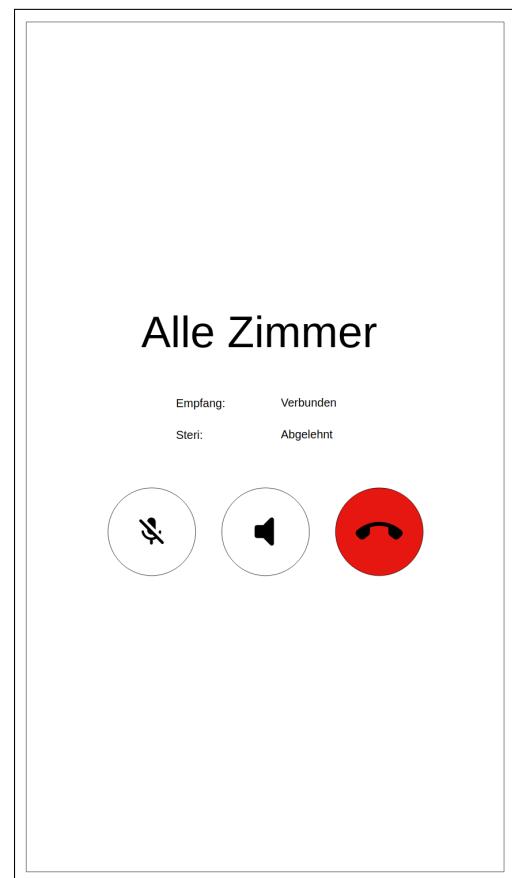
Nach dem Eingeben der Anmelddaten werden Praxismitarbeitende aufgefordert, die gewünschte Konfiguration auszuwählen. Die Ansicht (Abbildung 7.4) besteht aus einem Seitentitel und einer Liste zur Auswahl der gewünschten Konfiguration. In der Auswahl sind alle Zimmer zu sehen, welche dem Benutzer zur Verfügung stehen. Diese Konfigurationen müssen vor der Anmeldung im Admin UI erfasst und dem Benutzer zugewiesen werden. In der Kopfzeile sind die Schaltflächen "Zurück" und "Fertig" zu sehen. Die Schaltfläche "Zurück", bricht die Anmeldung ab und führt zurück zur Eingabe der Login-daten. Die Schaltfläche "Fertig" bestätigt die Auswahl und leitet zur Hauptansicht weiter. Wird bestätigt,

ohne dass ein Zimmer angewählt ist, wird dem Benutzer eine Fehlermeldung angezeigt und nicht zur Hauptansicht weitergeleitet.

Die Hauptansicht der Applikation gliedert sich in die Bereiche Home, Inbox und Einstellungen. Zwischen den drei Bereichen kann über eine Leiste am unteren Ende des Bildschirms navigiert werden. Die Ansicht Home zeigt dem Benutzer die Buttons, über welche er Benachrichtigungen versenden und Anrufe in der Gegensprechanlage starten kann. Wird ein Anruf gestartet, wird die Ansicht für aktive Anrufe angezeigt (Abbildung 7.6). Diese zeigt dem Benutzer den Titel des gestarteten Anrufs, sowie eine Liste aller Teilnehmer zusammen mit dem Verbindungsstatus jedes Teilnehmers. Der Titel des Anrufes entspricht dem Anzeigetext des verwendeten Buttons für ausgehende Anrufe und dem Namen des Anrufers für empfangene Anrufe. Neben den Anrufinformationen zeigt die Ansicht für aktive Anrufe drei Buttons. Über diese können Mikrofon und Lautsprecher des eigenen Gerätes stumm geschaltet werden. Weiter kann der Anruf über einen roten Button am rechten Rand beendet werden. Nach einem beendeten Anruf wird automatisch zu der zuvor angezeigten Ansicht navigiert.



**Abbildung 7.5:** Mockup Home



**Abbildung 7.6:** Mockup Aktiver Anruf

Der Bereich Inbox (Abbildung 7.7) zeigt eine Liste der empfangenen Benachrichtigungen sowie der empfangenen und verpassten Anrufe. Für alle Elemente in der Inbox wird der Name des Senders als Überschrift angezeigt. Darunter werden weitere Detailinformationen beschrieben. Für Benachrichtigungen beinhaltet dies den Textinhalt der Benachrichtigungen. Bei Anrufen beschrieben ob, es sich um einen empfangenen, verpassten oder abgelehnten Anruf handelt. Einträge für Benachrichtigungen sowie verpasste und abgelehnte Anrufe müssen durch eine Wischgeste quittiert werden. Die Funktionsweise der Quittierung wird aus dem bestehenden Mobile Client übernommen. Quittierte Meldungen werden aus der Inbox entfernt und nicht mehr angezeigt. Ein Quittieren von Meldungen und Anrufen passiert ausschliesslich lokal in der iOS Applikation. Der Empfänger wird nicht über die Quittierung informiert [2].

Es muss sichergestellt werden, dass verpasste Benachrichtigungen und Anrufe nicht übersehen werden. Dazu wird im Abstand von 60 Sekunden geprüft, ob unquittierte Benachrichtigungen oder Anrufe in der Inbox vorhanden sind. Ist dies der Fall, wird ein Erinnerungston abgespielt und eine Benachrichtigung angezeigt. Dieser Mechanismus wird in Kapitel 7.2.4 weiter beschrieben.



Abbildung 7.7: Mockup Inbox

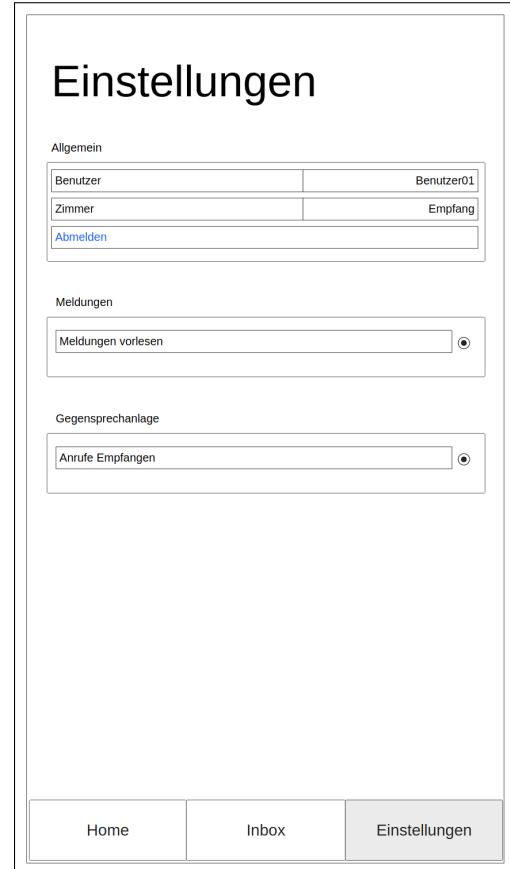


Abbildung 7.8: Mockup Einstellungen

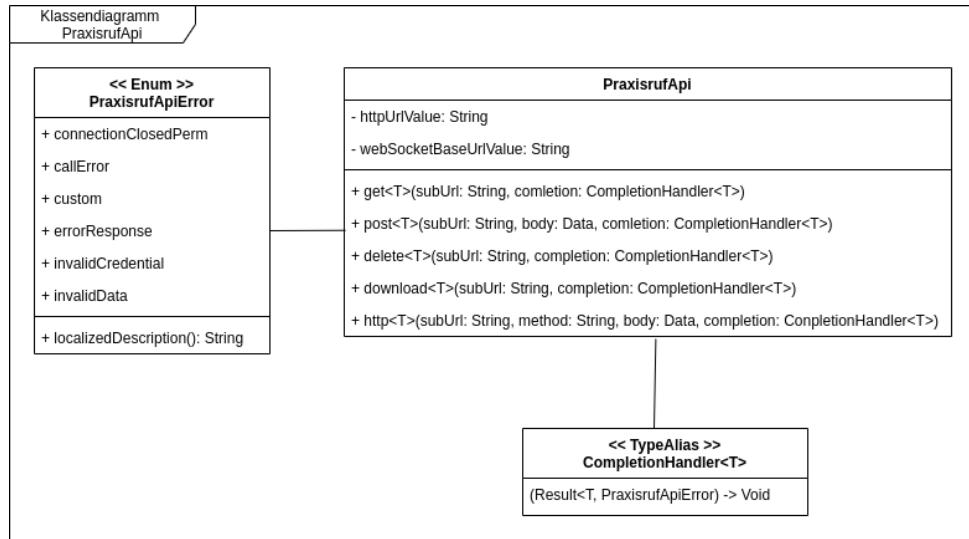
Abbildung 7.8 zeigt den Bereich Einstellungen. Der Bereich Einstellungen zeigt den aktuellen Benutzernamen und die gewählte Konfiguration. Über die Schaltfläche Abmelden, können sich Praxismitarbeiter aus der Applikation abmelden. Die Schaltfläche Benachrichtigungen vorlesen ist standardmäßig aktiviert. Wird die Option deaktiviert, werden Benachrichtigungen nie vorgelesen. Die Schaltfläche Anrufe empfangen ist ebenfalls standardmäßig aktiviert. Wird diese Option deaktiviert, werden alle empfangenen Anrufe automatisch abgelehnt und stattdessen eine Benachrichtigung angezeigt. Ausgehende Anrufe können auch getätigt werden, wenn diese Option aktiviert ist.

## 7.2.2 Anbindung Cloudservice API

Der Mobile Client muss an die API des Cloudservices angebunden werden. Es wird eine Anbindung an das Modul Configuration zur Anmeldung und Auswahl des gewünschten Zimmers und an das Modul Notification zum Versenden von Benachrichtigungen benötigt. Die Schnittstellen dieser Module stehen als Http-Endpunkte zur Verfügung. In diesem Unterkapitel wird beschrieben, wie Http-Anfragen an die Cloudservice API in den nativen iOS Client integriert werden. Das Abrufen von Sprachdaten und die Anbindung an die Signaling Instanz werden in den Kapiteln 7.3 und 7.4 beschrieben.

Die Basisbibliothek für iOS Entwicklung bietet die Klasse URLSession, über welche Netzwerkaufträge getätigkt werden können. Über URLSession.shared steht eine Standard-Instanz zur Verfügung, über

welche Netzwerkanfragen verarbeitet werden können [33]. Die Klasse UrlRequest ermöglicht es, HttpRequest für eine URL mit Header und Body zu erstellen [13]. Um die Integration dieser Klassen in die Applikation zu vereinfachen, wird ein zentraler Service mit dem Namen PraxisrufApi erstellt. Dieser kapselt das Erstellen, Befüllen und Absetzen der nötigen UrlRequest-Instanzen. Er bietet öffentliche Methoden für die Http-Verben Get, Post und Delete an. Über diese können Http-Requests mit der jeweiligen Methode abgesetzt werden. Zur Darstellung von Fehlern wird die Enum PraxisrufApiError erstellt. Diese definiert Fehlerkategorien und wird von PraxisrufApi verwendet, um Aufrufern das Fehlschlagen einer Anfrage mitzuteilen. Das Klassendiagramm in Abbildung 7.9 zeigt den Aufbau des Service PraxisrufApi.



**Abbildung 7.9:** Klassendiagramm PraxisrufApi

Die Basis-URL für Http-Anfragen wird in der Konfiguration der iOS Applikation definiert. Der PraxisrufApi Service lädt diese und verwendet sie für alle Abfragen, die abgesetzt werden. Alle öffentlichen Methoden von PraxisrufApi nehmen einen Parameter ”subUrl” als String entgegen. Dieser String wird der Basis-URL angehängt. Die Methoden Post nimmt zudem einen optionalen Parameter von Typ Data entgegen. Dieser definiert den Inhalt des Request Bodies. Mit diesen Informationen können Http-Requests erstellt und versendet werden.

Sämtliche öffentlichen Methoden des PraxisrufApi Service nehmen weiter einen Parameter mit dem Namen completion entgegen. Dabei handelt es sich um eine Funktion, welche beim Erfolg oder Fehlschlagen der Http-Anfragen ausgeführt wird. Als Parameter dieser Funktion wird immer der Typ Result<T, PraxisrufApiError> verwendet. Bei Result handelt es sich um einen Wrapper Typen welcher entweder das Resultat einer Abfrage oder ein Fehlerobjekt beinhaltet [34]. Im Fehlerfall wird das Result mit einem PraxisrufApiError Objekt erstellt. Im Erfolgsfall wird es mit den Daten aus dem Body der Http-Antwort befüllt. Zur Darstellung dieser Daten wird der generische Typ T verwendet. Der Typ wird vom Aufrufer von PraxisrufApi definiert und kann grundsätzlich beliebig sein. Er muss aber das Protokoll Decodable aus der iOS Standardbibliothek implementieren. Decodable Instanzen können von einer JSON-String-Repräsentation in ein Swift-Objekt konvertiert werden [35]. So kann das Resultat im Erfolgsfall aus der Response generiert und an die completion-Funktion übergeben werden. Dadurch kann die Konvertierung generisch in im PraxisrufApi-Service behandelt werden.

Anhand des Inhalts der Result-Instanz kann geprüft werden, ob die Anfrage erfolgreich war. Das Resultat kann entsprechend verarbeitet werden. Diese Prüfung und Verarbeitung findet innerhalb der completion-Funktion statt. Dieser Ansatz ermöglicht es im PraxisrufApi-Service ausschliesslich Http-Anfragen zu senden und die entsprechenden Antworten entgegenzunehmen. Der Api Service behandelt damit aus-

schliesslich die technische Anbindung an die API des Cloudservice. Er muss keine fachliche Logik implementieren und kann generisch für alle Anwendungen wiederverwendet werden.

Requests die über den PraxisrufApi-Service erstellt werden, werden automatisch authentisiert. Dazu lädt der Service das im KeyStore von iOS hinterlegte JWT Token. Das Token wird verwendet, um einen entsprechenden Authorization-Header zu generieren. Der generierte Authorization Header wird der Http-Anfrage angefügt. Ist kein Token vorhanden, wird keine Anfrage abgesetzt. Es wird direkt die completion-Funktion mit dem Fehler PraxisrufApiError.invalidCredential aufgerufen.

Mit dieser Lösung steht ein Service zur Verfügung, über welchen Http-Anfragen einfach in eine iOS App integriert werden können. Dank der generischen Methoden im PraxisrufApi-Service können neue Calls einfach hinzugefügt werden, ohne das Boilerplate-Code wiederholt werden muss. Durch die completion-Funktionen kann die fachliche Verarbeitung von Http-Anfragen vom Aufrufer definiert werden.

Um die Verwendung der Cloudservice API weiter zu vereinfachen, wird der PraxisrufApi-Service um spezifische Methoden für benötigte Abfragen erweitert. Dazu werden mehrere Extension-Klassen erstellt. Diese definieren Methoden mit sprechenden Namen für die benötigte Funktionalität und kapseln die Verwendung der Get, Post und Delete Methoden.

Der PraxisrufApi-Service ermöglicht es, Abfragen an die Cloudservice API abzusetzen. Die Resultate dieser Abfragen müssen in der Benutzeroberfläche angezeigt werden können. Es wird pro Domäne ein weiterer Service geschrieben, welche den Aufruf des PraxisrufAPI-Services kapselt. Dieser Service bietet Methoden, über welche der PraxisrufApi-Service angesprochen werden kann. View-Komponenten können diese Services nutzen, um durch Benutzereingaben ausgelöste Anfragen an die Cloudservice Api zu senden. Resultate und Fehler aus Anfragen an den Cloudservice werden in diesen Services als Instanzvariablen gehalten. Die View-Komponenten können lesend auf diese Variablen zugreifen, um die entsprechenden Resultate oder Fehler anzuzeigen.

### 7.2.3 Anbindung Messaging Service

Um Benachrichtigungen empfangen zu können, muss Firebase Cloud Messaging an die native iOS Applikation angebunden werden. Firebase bietet eine Bibliothek mit welcher Firebase Cloud Messaging in iOS Clients integriert werden kann [11]. Diese Integration kann allerdings nicht mit dem Mitteln von SwiftUI implementiert werden. Dies liegt daran, dass für das Empfangen von Benachrichtigungen und das Anzeigen von Push-Benachrichtigungen Integration mit dem Benachrichtigungszenter des Betriebssystems notwendig ist. Diese Integration kann bis heute nur über AppDelegate umgesetzt werden. Zur Anbindung von Firebase Cloud Messaging wird dementsprechend ein AppDelegate implementiert. Dieser muss bei Initialisierung der SwiftUI Applikation registriert werden.

Die Logik des AppDelegate wird auf das minimal Nötige reduziert. Der AppDelegate selbst ist für die direkte Kommunikation mit Firebase verantwortlich und muss empfangene Daten an Betriebssystem und SwiftUI Applikation übergeben. Fachliche Logik wird nicht im AppDelegate, sondern in der SwiftUI Applikation ausgeführt. Dies ermöglicht es die Anbindung des Messaging Service im AppDelegate zu kapseln. Sollte Firebase Cloud Messaging in Zukunft durch einen anderen Anbieter ersetzt werden, muss damit ausschliesslich die Logik im AppDelegate angepasst werden. Diese Trennung stellt sicher, dass die Fachlogik vollständig mit SwiftUI implementiert werden kann. Der AppDelegate beinhaltet lediglich die Teile, welche aus technischen Gründen nicht mit SwiftUI umgesetzt werden können.

Um Benachrichtigungen von Firebase Cloud Messaging empfangen zu können, muss der AppDelegate folgende Funktionalität umsetzen. Beim Start der Applikation muss sich der Mobile Client beim Messaging Service registrieren. Nach der Registrierung wird für den Mobile Client ein Token generiert, welches den Client eindeutig beim Messaging Service identifiziert. Der AppDelegate muss, darauf reagieren und das erneuerte Token an die Applikation übergeben.

Für die Verarbeitung von Benachrichtigungen muss der AppDelegate Benachrichtigungen im Vordergrund empfangen und dem Betriebssystem zur Anzeige übergeben. Die Informationen aus der empfangenen Benachrichtigung müssen anschliessend an die Applikation übergeben werden. Benachrichtigungen, die im Hintergrund empfangen werden, müssen an das Betriebssystem übergeben und angezeigt werden. Sobald die Applikation wieder in den Vordergrund tritt, müssen die Daten an die Applikation zur weiteren Verarbeitung übergeben werden.

#### 7.2.4 Benachrichtigungen prüfen

Der bestehende Mobile Client prüft in regelmässigen Abständen, ob ungelesene Benachrichtigungen in der Inbox vorhanden sind. Wenn ungelesene Benachrichtigungen gefunden werden, wird eine Benachrichtigung angezeigt und ein Ton abgespielt. Im Mobile Client der Vorgängerlösung findet diese Prüfung nur statt, wenn die Applikation in Vordergrund aktiv ist. Diese Funktion wird in der nativen iOS Applikation übernommen.

Für die Umsetzung der Erinnerungsfunktion werden zwei Services definiert. Erstens wird eine Inbox erstellt, welche eine Liste der aktuellen Benachrichtigungen führt. Zweitens wird ein InboxReminderService implementiert. Dieser prüft den Inhalt der Inbox und sucht nach unquittierten Elementen, welche älter als eine Minute sind. Werden solche Elemente gefunden, wird eine Benachrichtigung angezeigt und ein Benachrichtigungston abgespielt. Die regelmässige Prüfung der Inbox wird mit der Timer-Klasse der iOS Standardbibliothek umgesetzt. Über diese ist es möglich auf einer View in regelmässigen Abständen Events auszulösen [14]. Ein solcher Timer wird auf der Hauptansicht für angemeldete Benutzer registriert. Der Timer löst alle 60 Sekunden die Prüfung des InboxReminderService aus.

Die Prüfung von Benachrichtigungen im Hintergrund wird im Rahmen dieses Projektes nicht umgesetzt. Für künftige Erweiterungen ist es möglich diese Funktion zu implementieren. Um dies zu ermöglichen, müssen Benachrichtigungen auf dem Gerät persistiert werden. So stehen die Daten auch zur Verfügung, wenn die Applikation nicht gestartet ist. Weiter muss ein Hintergrundtask implementiert und registriert werden [15], welcher die persistierten Daten lädt und die darauf die Prüfung des InboxReminderService ausführt.

#### 7.2.5 Security

In einem Praxisrufsystem muss die sichere Übertragung von Daten gewährleistet sein. Die dazu definierten Konzepte werden aus dem Vorgängerprojekt übernommen.

Alle Daten zwischen den Services müssen über verschlüsselte Verbindungen ausgetauscht werden. Http-Anfragen CloudService API erfolgen ausschliesslich über Https. Alle Anfragen an die API des Cloudservice müssen zudem mit einem Json Web Token (JWT) authentifiziert sein. Praxismitarbeitende werden durch den Cloudservice mittels Basic Authentication authentifiziert. Bei der Anmeldung mit Benutzernname und Passwort liefert der Cloudservice ein JWT, welches vom Client für weitere Anfragen verwendet wird. Sowohl die Credentials für die Basic Authentication als auch das JWT Token werden durch die iOS Applikation im Keystore des Betriebssystems gespeichert [2]. Das vom Cloudservice ausgestellte Token hat eine Gültigkeitsdauer von 24h und wird alle 12h erneuert. Dazu wird die Basic Authentication mit den gespeicherten Credentials wiederholt. Die dazu nötige Http-Anfrage wird über eine weitere Timer-Instanz auf der Hauptansicht für angemeldete Benutzer ausgelöst. Der Ablauf für Authentifizierung und Authorisierung wird damit unverändert aus dem Vorgängerprojekt übernommen. Die entsprechenden Abläufe sind in den Kapiteln 5.3.6 und 5.3.7 im Projektbericht "IP5 Cloudbasiertes Praxisrufsystem" dokumentiert [2].

### 7.2.6 Servicemodell

Dieses Kapitel gibt einen Überblick zu den Services welche in der nativen iOS Applikation implementiert werden. Abbildung 7.10 zeigt das vollständige Modell der implementierten Services. Diese Darstellung beinhaltet die Services, welche in den Kapiteln 7.2 bis 7.4 beschrieben werden. View- und Model-Komponenten werden hier nicht dargestellt.

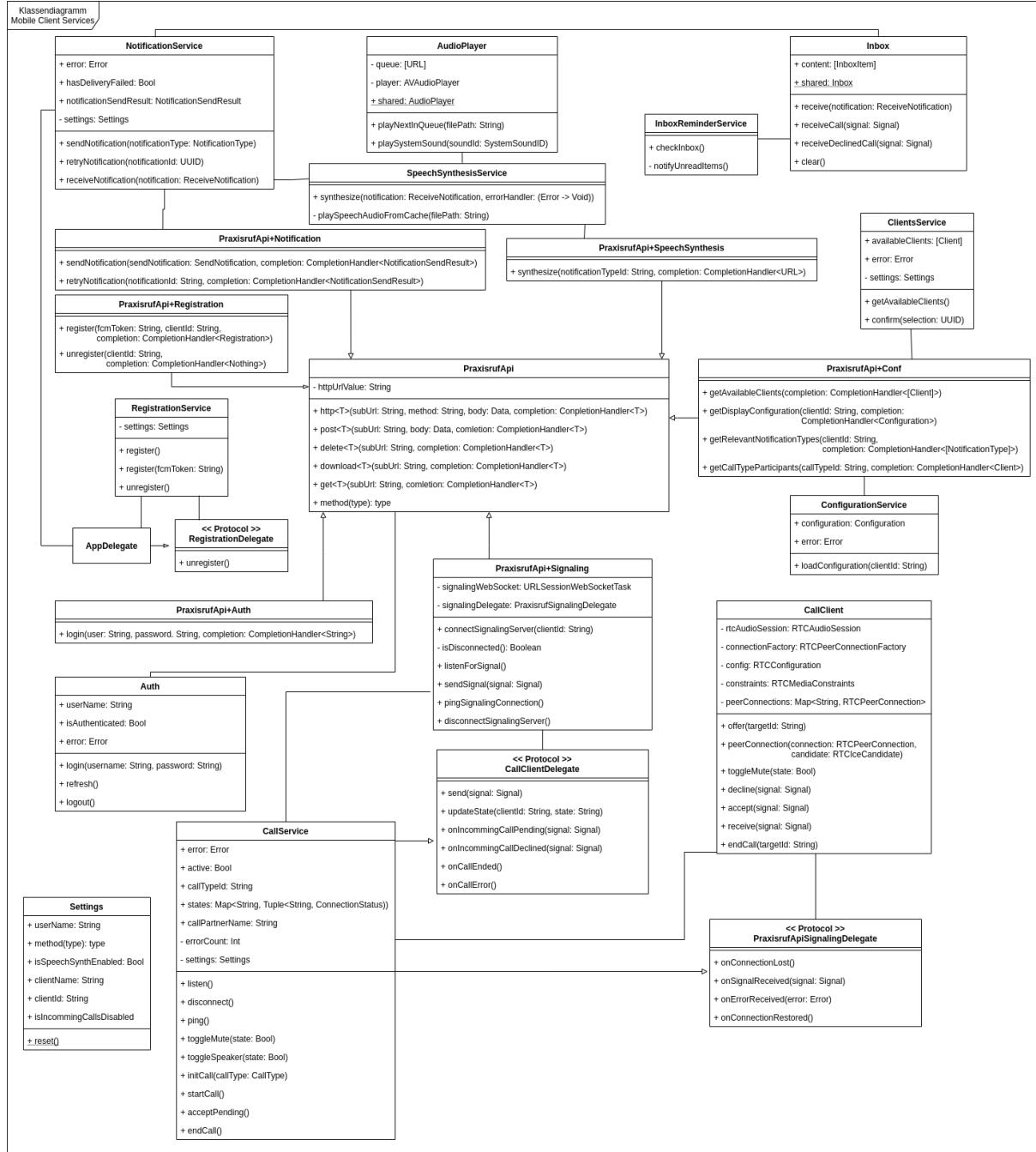


Abbildung 7.10: Klassendiagramm - Mobile Client Services

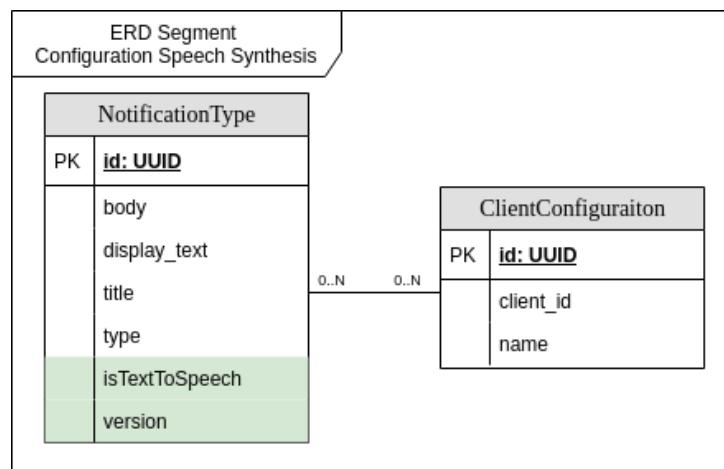
Die Klasse `Settings` wird verwendet um die Konfiguration des Benutzers zu verwalten. Sie bietet Getter und Setter für alle Properties. Nach dem setzen eines Wertes, wird dieser über die Komponente `UserData` aus der iOS Standardbibliothek persistiert. Die Verwendungen der `Settings`-Klasse sind in Abbildung 7.10 nicht dargestellt, um die Übersichtlichkeit zu wahren.

## 7.3 Sprachsynthese

Dieses Kapitel beschreibt die Integration von Sprachsynthese in Praxisruf. Der Fokus liegt dabei auf den Abläufen zum Empfangen von Benachrichtigungen und dem Abrufen der Sprachdaten. Der Empfang von Benachrichtigungen wird so erweitert, dass der Inhalt empfangener Benachrichtigungen automatisch vorgelesen wird.

### 7.3.1 Konfiguration

Benachrichtigungen für Praxisruf können über das Admin UI konfiguriert werden. Es kann pro Benachrichtigung Titel, Inhalt, Anzeigetext für Benachrichtigungsbuttons und eine Beschreibung erfasst werden. Diese Konfiguration wird über die Entität NotificationType verwaltet. Neu soll auch konfiguriert werden können, ob eine Benachrichtigung für die Sprachsynthese relevant ist. Dazu wird die Entität NotificationType um ein boolean Flag mit dem Namen "isTextToSpeech" erweitert. Dieses Flag wird beim Versenden einer Benachrichtigung mitgesendet und kann vom Empfänger überprüft werden. Wenn das Vorlesen von Benachrichtigungen in den lokalen Einstellungen und das Flag auf der Benachrichtigung aktiviert sind, wird die Benachrichtigungen vorgelesen. Abbildung 7.11 zeigt einen Ausschnitt aus dem Entity Relationship Diagramm der Domäne Configuration. Dabei sind die Felder, welche für die Sprachsynthese ergänzt werden, grün markiert.



**Abbildung 7.11:** ERD Ausschnitt - Konfiguration Sprachsynthese

Neben dem Feld isTextToSpeech, wird die NotificationType Entity um ein weiteres Feld "version" erweitert. Das Versionsfeld beinhaltet eine Ganzzahl, welche mit jeder Änderung inkrementiert wird. Der Inhalt dieses Felds wird ebenfalls beim Versenden von Benachrichtigungen mitgesendet. Auf Client-Seite wird diese Information zur Implementierung eines Cache verwendet.

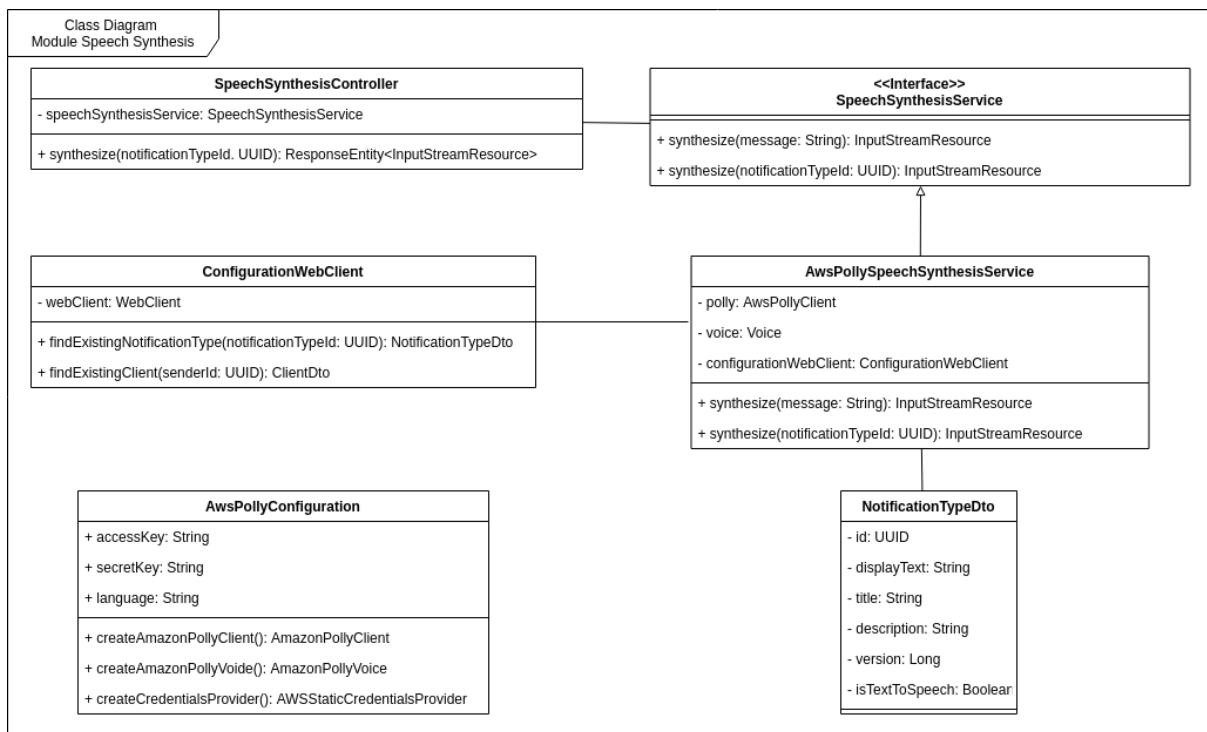
### 7.3.2 Anbindung von Sprachsynthese in Cloudservice

Dieses Kapitel beschreibt, wie Amazon Polly an den Cloudservice angebunden wird, um das Vorlesen von Benachrichtigungen zu ermöglichen.

Die Anbindung an Amazon Polly erfolgt zentral im Cloudservice. Sämtliche Anfragen an Amazon Polly werden durch den Cloudservice gemacht. Empfänger von Benachrichtigungen senden keine direkten Anfragen an Amazon Polly. Sie kommunizieren stattdessen mit dem Cloudservice. Dieser führt die Anfrage an Amazon Polly aus und gibt die Resultate an den Anfrager zurück.

Für die Anbindung von Amazon Polly wird der Cloudservice um ein Modul mit dem Namen "Speech Synthesis" erweitert. Dieses Modul muss unabhängig von allen anderen Domänen-Modulen des Cloudservice umgesetzt werden. Werden Daten aus einer anderen Domäne benötigt, muss die Kommunikation über die API des entsprechenden Moduls gehen. Diese Trennung ermöglicht es, das Modul in Zukunft einfach aus dem Cloudservice auszubauen und als eigenständigen Microservice zu betreiben.

Die Abhängigkeit zu Amazon Polly als Anbieter soll weitmöglichst minimiert werden. So kann bei Bedarf einfacher auf einen anderen Provider gewechselt werden. Um dies zu ermöglichen wird das Interface SpeechSynthesisService definiert. Dieses gibt eine einzelne Methode vor, welche eine InputStreamResource zurückgibt und eine Universal Unique Id (UUID) als Parameter entgegennimmt. Der Parameter entspricht der technischen Identifikation des zu synthetisierenden Benachrichtigungstypes (NotificationType). Die InputStreamResource muss die synthetisierten Sprachdaten enthalten. Dieses Interface wird von der Komponente, welche die Schnittstelle nach aussen bietet verwendet. Um einen Anbieter für Sprachsynthese anzubinden, kann dieses Interface implementiert werden und der Schnittstelle zur Verfügung gestellt werden. Das Klassendiagramm in Abbildung 7.12 gibt einen Überblick über den Aufbau des Moduls Speech Synthesis.



**Abbildung 7.12:** Klassendiagramm - Modul SpeechSynthesis

Für die Anbindung des Providers Amazon Polly wird das Interface SpeechSynthesisService mit der Klasse AwsPollySpeechSynthesisService implementiert. Amazon stellt einen Java Bibliothek für Amazon Polly zur Verfügung, welcher diese Anbindung ermöglicht [18]. Diese Bibliothek bietet alle Klassen die für die Anbindung an AWS Polly nötig sind und wird in der Implementierung des SprachSynthese-ProviderService verwendet, um den Service anzubinden.

Die Anbindung von Amazon Polly benötigt drei Komponenten. Als Erstes muss eine Instanz von AWSStaticCredentialsProvider zur Verfügung gestellt werden. Dieser liefert die Credentials, welche das System berechtigen, Anfragen an AWS Polly zu senden. Als Zweites muss eine Voice konfiguriert werden. Die Voice definiert Sprache und Stimme, welche für die Sprachausgabe verwendet wird. Letztlich muss eine Instanz von AmazonPollyClient konfiguriert werden. Dieser Client wird verwendet, um Anfragen an AWS Polly zu senden. Er verwendet den zuvor konfigurierten CredentialsProvider, um die Anfrage

mit den entsprechenden Credentials zu ergänzen. Die zuvor konfigurierte Voice wird bei Anfragen an Amazon Polly mitgesendet, damit die Daten mit den gewünschten Parametern synthetisiert werden.

Der Cloudservice ist als Java-Applikation mit Spring Boot umgesetzt. Dies ermöglicht es, die notwendigen Komponenten in einer Spring Konfigurationsklasse zu konfigurieren und als Spring Beans zu instanzieren. Über die Dependency Injection von Spring werden diese Komponenten dem AwsPollySpeechSynthesisService übergeben werden.

Werte, welche für die technische Konfiguration notwendig sind, werden aus der Konfigurationsdatei application.yml geladen. Sprache und Region werden sich im Rahmen dieses Projektes nie ändern und beinhalten keine sensitiven Informationen. Sie werden deshalb direkt in der Konfigurationsdatei definiert und mit dem Quellcode des Projektes verwaltet. Als Credentials für die Anbindung dienen die zwei Schlüssel AccessKey und SecretKey. Credentials werden nicht direkt in der Konfigurationsdatei gespeichert. Stattdessen wird ein Platzhalter definiert, welcher die Werte für Credentials aus entsprechend benannten Umgebungsvariablen lädt. Die Zugangsdaten müssen damit nicht mit dem Quellcode verwaltet werden.

### 7.3.3 Sprachsynthese über Cloudservice API

Endgeräte in Praxisruf müssen Sprachdaten über den Cloudservice beziehen können. Das Modul Speech Synthesis stellt deshalb eine Schnittstelle zur Verfügung, über welche Sprachdaten abgefragt werden können. Dabei ist es nicht möglich beliebige Textdaten in Sprachdaten zu verwandeln. Stattdessen erlaubt die Schnittstelle die Abfrage von Sprachdaten für Inhalt und Sender einer Benachrichtigung.

Als Inhalt einer Benachrichtigung wird das Feld "title" aus der Entität NotificationType verwendet. Der Name des Senders wird dem Feld "name" der Entität Client entnommen. Beide Entitäten sind Teil des Moduls Configuration. Die entsprechenden Daten müssen deshalb über die API des Configuration-Moduls geladen werden. Um dies zu ermöglichen, werden die Identifikatoren der relevanten Entitäten zusammen mit Benachrichtigung versendet.

Der Endpunkt zum Bezug von Sprachdaten nimmt die zwei Parameter "notificationTypeId" und "sender" entgegen. Diese müssen die technischen Identifikatoren der jeweiligen Entitäten beinhalten. Anhand dieser Parameter werden die benötigten Daten von der API des Configuration-Moduls geladen. Anschliessend wird eine Anfrage an Amazon Polly gesendet um die Textdaten als Sprache zu synthetisieren. Der zu synthetisierende Text setzt sich dabei aus Inhalt der Benachrichtigung und Name des Senders zusammen. Die beiden Werte werden dabei durch ein Komma getrennt. Dadurch wird eine Pause zwischen dem Vorlesen der einzelnen Werte eingefügt. Die von Polly gelieferten Sprachdaten können anschliessend als Resultat zurückgegeben werden.

Der Endpunkt für die Abfrage von Sprachdaten im Cloud Service wird als Spring RestController umgesetzt. Die Sprachdaten werden darin als Binärdaten mit Media Type "audio/mp3" im Body der Response zurückgegeben. Der Endpunkt wird kann über Http-Get-Anfragen angesprochen werden.

### 7.3.4 Security

Anfragen an die API des Moduls Speech Synthesis müssen, wie alle Anfragen an die Cloudservice API, authentisiert werden. Für die Authentisierung wird derselbe Mechanismus wie für Http-Anfragen in allen Cloudservice Modulen verwendet. Über die Konfiguration des App Modules des Cloudservices wird die Authentifizierung aller Http-Requests überprüft. Mit dieser Prüfung wird sichergestellt, dass ein gültiges JWT Token im Authentication Header der Anfrage vorhanden ist [2]. Diese Prüfung wurde im Rahmen des Vorgängerprojektes umgesetzt und wird weiterverwendet. Die entsprechenden Abläufe sind in den Kapiteln 5.3.6 und 5.3.7 im Projektbericht "IP5 Cloudbasiertes Praxisrufsystem" dokumentiert [2].

Um die Verschlüsselung der Übertragung von Sprachdaten und Anfragen zwischen Cloudservice und Mobile Client wird für die Übertragung ausschliesslich das Protokoll HTTPS verwendet. Die Übertragung von Daten zwischen Cloudservice und Amazon Polly ist über Secure Sockets Layer (SSL) geschützt [36].

### 7.3.5 Sprachsynthese in iOS App

In der iOS App müssen empfangene Benachrichtigungen vorgelesen werden können. Um dies zu ermöglichen wird eine Anbindung an die Sprachsynthese-API des Cloudservice umgesetzt. Dazu wird die in Kapitel 7.2 beschriebene Klasse PraxisrufApi erweitert. Neben dem Abfragen von JSON Daten über HTTP Schnittstellen, muss diese für die Sprachsynthese auch das Herunterladen von Dateien unterstützen. Dazu wird die Komponente URLSession aus der iOS Standardbibliothek verwendet. Diese bietet mit URLSession.downloadTask die Möglichkeit Inhalte von einer URL herunterzuladen [37].

Der Service PraxisrufApi wird um eine Methode mit dem Namen "download" ergänzt. Diese ist dafür verantwortlich, eine Anfrage für den Download mit Credentials aus dem iOS Keystore zu ergänzen und die Anfrage zu versenden. Die Resultate der Anfrage und aufgetretene Fehler werden analog zu anderen Abfragen an eine Callback-Funktion übergeben. Heruntergeladene Dateien werden von PraxisrufApi in einem temporären Verzeichnis gespeichert. Das Resultat im Erfolgsfall ist deshalb nicht die heruntergeladene Datei selbst, sondern eine URL welche auf die Datei im temporären Verzeichnis zeigt.

Die Sprachsynthese für Benachrichtigungen muss automatisch ausgeführt werden, nachdem eine relevante Benachrichtigung empfangen wurde. Der Empfang der Benachrichtigung findet über die Anbindung von Firebase Cloud Messaging im AppDelegate statt. Die Benachrichtigung wird im AppDelegate empfangen und an die Applikation übergeben. Die empfangene Benachrichtigung beinhaltet mit dem "isTextToSpeech" Flag, die Information, ob sie für die Sprachsynthese relevant ist.

Ist eine Benachrichtigung für Sprachsynthese relevant, werden die Sprachdaten dazu vom Cloudservice bezogen. Dazu wird ein SpeechSynthesisService implementiert, welcher PraxisrufApi verwendet, um eine Anfrage an den Cloudservice zu senden. Wurden die Daten erfolgreich geladen, kopiert der SpeechSynthesisService die heruntergeladenen Daten aus dem temporären Downloadverzeichnis in ein permanentes Verzeichnis. Die Datei wird dabei unter dem Namen *NotificationTypeId.Version.SenderId* gespeichert. Sowohl NotificationTypeId als auch Version und SenderId können der empfangenen Benachrichtigung entnommen werden. Nachdem die Sprachdatei unter dem neuen Namen gespeichert ist, wird ihr Inhalt abgespielt.

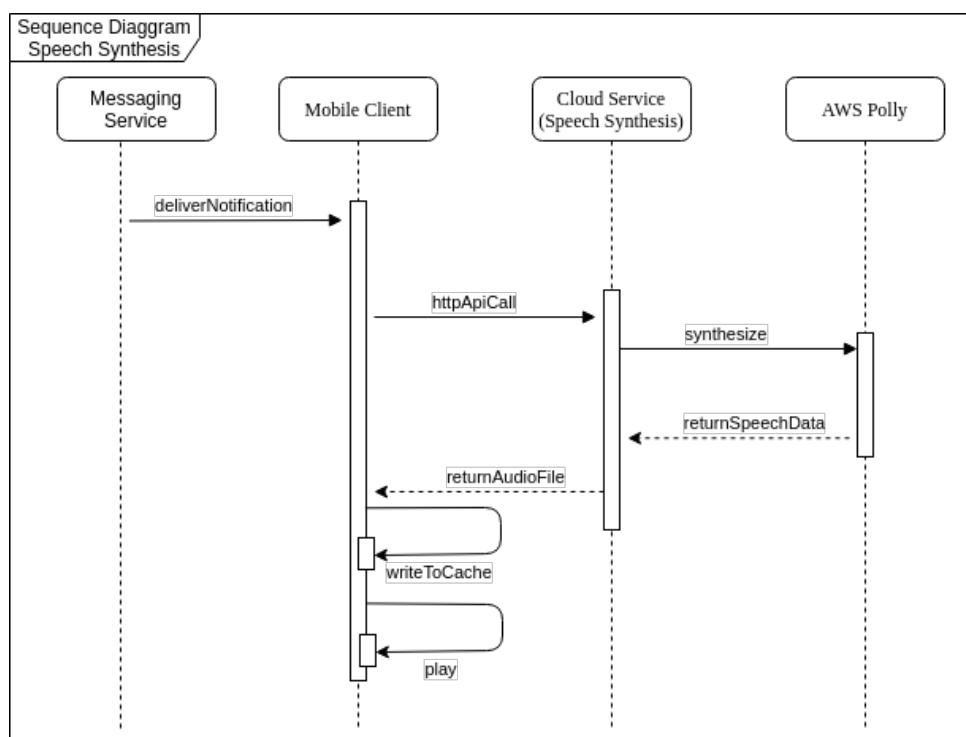
Die Namenskonvention für die gespeicherten Sprachdateien, erlaubt es ein Cache auf der Seite der iOS Applikation umzusetzen. Bevor der SpeechSynthesisService eine Anfrage an den Cloudservice absetzt, prüft er, ob bereits eine Datei mit dem entsprechenden Namen vorhanden ist. Ist dies der Fall, wird keine Anfrage an den Cloudservice gesendet und es wird die bereits gespeicherte Sprachdatei abgespielt. Dieses Cache ermöglicht es Anfragen für Sprachsynthese zu minimieren und nach Änderungen trotzdem immer die aktuellsten Daten zu erhalten.

### 7.3.6 Laufzeitsicht

Dieses Kapitel beschreibt die Prozesse für das Vorlesen von Benachrichtigungen. Dabei wird der Ablauf vom Versenden der Benachrichtigung bis hin zur Ausgabe der Sprachdaten auf Empfängerseite beschrieben. Abbildung 7.13 stellt den Ablauf dem Empfangen einer Benachrichtigung aus Systemsicht dar.

Um eine Benachrichtigung zu versenden, sendet ein Mobile Client eine Anfrage an den Cloudservice. Dieser lädt die gespeicherte Konfiguration und findet alle für die gewünschte Benachrichtigung relevanten Empfänger. Anschliessend erstellt er für jeden Empfänger eine Benachrichtigung und versendet diese über den Messaging Service. Der Messaging Service stellt die Benachrichtigungen an die Empfänger zu [2].

Benachrichtigungen werden im Mobile Client über die Anbindung an den Messaging Service im App-Delegate empfangen. Im AppDelegate werden die Informationen aus der empfangenen Benachrichtigung gelesen und in das interne Model der Mobile Client Applikation überführt. Anschliessend wird die Benachrichtigung an das Betriebssystem übergeben damit auf dem Gerät ein Benachrichtigungston abgespielt und eine Push-Benachrichtigung angezeigt wird. Daraufhin wird die Benachrichtigung im internen Model einem NotificationService übergeben. Dieser fügt die empfangene Benachrichtigung in eine Inbox ein. Ab diesem Moment ist die Benachrichtigung in der Inbox des Mobile Clients ersichtlich.



**Abbildung 7.13:** Sequenzdiagramm - Sprachsynthese auf Systemebene

Nachdem eine empfangene Benachrichtigung der Inbox hinzugefügt wurde, wird geprüft ob Sprachsynthese in den lokalen Einstellungen aktiviert ist. Ist diese deaktiviert, endet die Verarbeitung. Andernfalls wird geprüft, ob das "isTextToSpeech" Flag auf der Benachrichtigung aktiviert ist. Nur wenn das Flag aktiviert ist, wird die Benachrichtigung an den SpeechSynthesisService übergeben. Der SpeechSynthesisService prüft als erstes, ob die Sprachdaten für die empfangene Benachrichtigung bereits lokal zur Verfügung stehen. Dies wird gemacht in dem er überprüft, ob im Applikationsverzeichnis bereits eine Datei für Id, Version und Sender der Benachrichtigung vorhanden ist. Ist dies der Fall, werden die Inhalte dieser Datei abgespielt und es wird keine Anfrage an den Cloubservice versendet. Wenn die Daten gar

nicht oder nur in einer anderen Version lokal gefunden werden, wird eine Anfrage an den CloudService gesendet.

Sobald Sprachdaten über die Cloudservice API angefragt werden, lädt dieser den Namen des Senders und die Inhalte der Benachrichtigung aus der Konfiguration. Anschliessend sendet der Cloudservice eine Anfrage an Amazon Polly, um den Titel der Benachrichtigung als Sprachdaten zu synthetisieren. Die Resultate von Amazon Polly werden als Resultat der Anfrage des Mobile Clients zurückgegeben. Der Client speichert die empfangenen Daten lokal im Applikationsverzeichnis unter Id und Version verknüpften NotificationType. Nachdem die Daten gespeichert wurden, wird deren Inhalt abgespielt.

## 7.4 Gegensprechanlage

Mit der Integration von synchroner Sprachübertragung wird das Praxisrufsystem um die Funktion Gegensprechanlage erweitert. Die gewählte Technologie WebRTC erlaubt es, Sprachverbindungen zwischen Clients aufzubauen. Dieses Kapitel beschreibt wie das Praxisrufsystem erweitert wird, um eine konfigurierbare Gegensprechanlage mit WebRTC zu implementieren.

### 7.4.1 Konfiguration

Die Gegensprechanlage wird in den nativen Mobile Client integriert. Praxismitarbeitende können über Buttons Sprachverbindungen zu anderen Clients aufbauen. Welche Buttons und damit welche Sprachverbindungen zur Verfügung stehen, wird durch Praxisadministrierende über das Admin UI konfiguriert. Damit dies möglich ist, sind Änderungen an de Configuration-Modul des Cloudservice sowie am Admin UI notwendig.

Die Konfiguration von Mobile Clients wird in der Domäne Configuration abgebildet. Zentral sind dabei die beiden Entities Client und ClientConfiguration. Ein Client repräsentiert ein physisches Endgerät. Eine ClientConfiguration definiert die Konfiguration eines Gerätes.

Praxisruf bietet bereits heute die Möglichkeit Buttons zu konfigurieren, über welche Benachrichtigungen versendet werden können. Diese Buttons werden mit der Entität NotificationType konfiguriert, welche wiederum einer ClientConfiguration zugeordnet werden können. Diese ClientConfiguration wird bei der Anmeldung auf dem Mobile Client geladen und verwendet, um die nötigen Buttons darzustellen. Für die Konfiguration von Sprachverbindungen wird die Entität CallType erstellt. Ein CallType beinhaltet den Text, welcher auf dem zugehörigen Button auf Clientseite angezeigt wird und eine Liste von Clients, welche als Ziel der Sprachverbindung verwendet werden. Abbildung 7.14 zeigt einen Ausschnitt aus dem Entity Relationship Diagramm der Configuration Domäne. Dabei sind die Teile, die für die Konfiguration von Sprachverbindungen ergänzt werden, grün markiert.

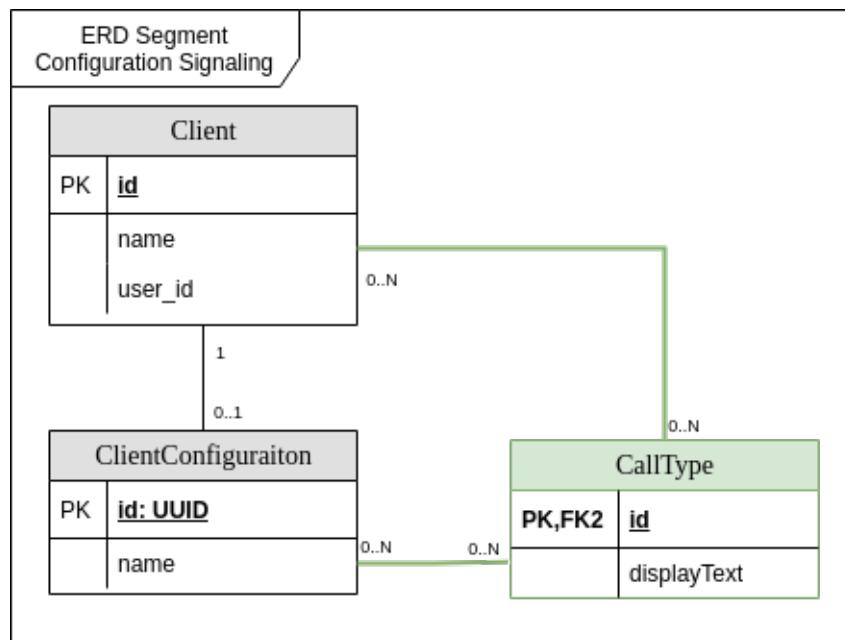


Abbildung 7.14: ERD Ausschnitt - Konfiguration Gegensprechanlage

Das Admin UI wird mit Ansichten erweitert, über welche CallTypes erstellt, angezeigt, bearbeitet und gelöscht werden können. Gleichzeitig wird der Cloudservice um Rest Endpunkte für das Lesen, Erstellen, Aktualisieren und Löschen von CallTypes erweitert. Die Ansichten für ClientConfigurations im Admin

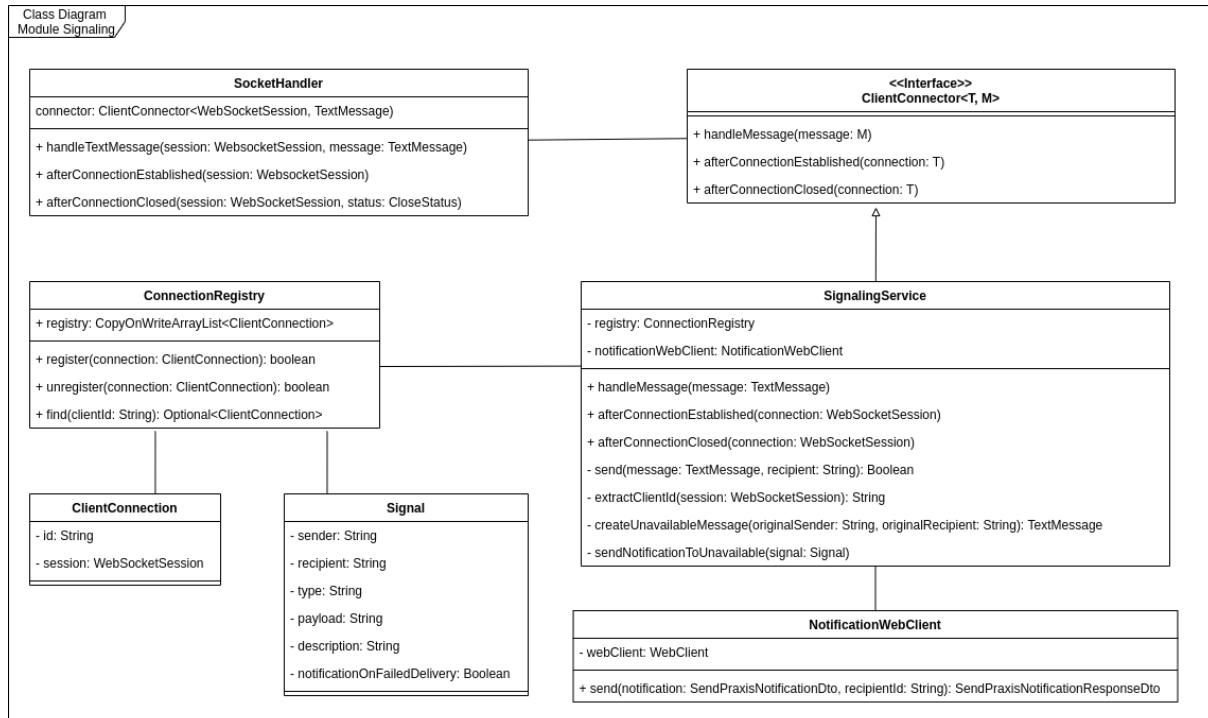
UI werden so erweitert, dass CallTypes darauf angezeigt, hinzugefügt und entfernt werden können. Die API des Cloudservice wird erweitert, um die erweiterte Konfiguration verwalten zu können.

### 7.4.2 Signaling Instanz

Mit WebRTC werden Peer-To-Peer Verbindungen aufgebaut [4]. Damit diese Verbindungen aufgebaut werden können, müssen die beteiligten Geräte Signalmeldungen austauschen können. Dazu ist eine Instanz notwendig, welche Signale zwischen den Endgeräten vermitteln kann. Diese Signaling Instanz wird als Teil des Cloudservice implementiert. Abbildung 7.15 zeigt das Klassendiagramm der dazu umgesetzten Komponenten.

Der Cloudservice wird um ein neues Modul "Signaling" erweitert. Dieses soll den Austausch von Signalen zwischen Clients ermöglichen. Gleich wie das Modul für Sprachsynthese wird es unabhängig von den anderen Domänenmodulen im Cloudservice implementiert. Das Modul Signaling muss dabei zwei Aufgaben übernehmen. Erstens muss es Mobile Clients die Möglichkeit bieten, sich für Sprachverbindungen zu registrieren. Zu diesem Zweck müssen Mobile Clients eine Verbindung mit der Signaling Instanz herstellen und trennen können. Zweitens muss es Signalmeldungen empfangen und an die relevanten Empfänger zustellen können. Kann eine Signalmeldung nicht zugestellt werden, muss es den betroffenen Empfänger über das verpasste Signal informieren.

Für diese Funktionen wird das Interface ClientConnector definiert. Dieses definiert die Methoden afterConnectionEstablished und afterConnectionClosed. Die beiden Methoden werden aufgerufen, wenn eine Verbindung geöffnet bzw. geschlossen wurde. Die Implementierung dieses Interfaces ist dafür verantwortlich verfügbare Verbindungen zu verwalten. Weiter definiert das Interface die Methode handleSignal. Diese muss verwendet werden, um ein Signal entgegenzunehmen und an relevante Empfänger weiterzuleiten.



**Abbildung 7.15:** Klassendiagramm - Modul Signaling

Für die Verwaltung von Verbindungen wird die Komponente ConnectionRegistry implementiert. Diese führt eine Liste bekannter Verbindungen und bietet Methoden um Verbindungen zu registrieren und wieder entfernen. Sie bietet weiter eine Methode, um zu überprüfen, ob eine bestimmte Verbindung bekannt

ist. Bekannte Verbindungen werden mit der Klasse ClientConnection abgebildet. Ein ClientConnection beinhaltet immer einen Identifikator und das technische Verbindungsobjekt. So können Verbindungen in der ClientConnection immer über einen eindeutigen Identifikator registriert und gefunden werden.

Die Klasse SignalingService implementiert das ClientConnector Interface. Sie verwendet die Klasse ConnectionRegistry, um eine Liste von verfügbaren Verbindungen zu führen. Über die Methoden onConnectionEstablished werden neue Verbindungen in der ConnectionRegistry registriert. Mit der Methode onConnectionClosed werden geschlossene Verbindungen wieder entfernt.

Für das Zustellen von Signalen über bekannte Verbindungen wird die Methode handleSignal implementiert. Jedes Signal beinhaltet die Identifikation seines Empfängers. Beim Empfang eines Signals wird kontrolliert, ob die ConnectionRegistry eine Verbindung für die Identifikation des Empfängers enthält. Ist dies der Fall, wird das Signal über diese Verbindung an den Empfänger übermittelt. Wenn dies nicht der Fall ist oder wenn das Senden des Signals fehlschlägt, ist der Empfänger nicht erreichbar. Nicht erreichbare Empfänger werden mit Benachrichtigungen über verpasste Signale informiert. Dazu wird eine Benachrichtigung über die API des Moduls Notification versendet.

Die Schnittstelle der Signaling Instanz im Cloudservice wird mit Websockets umgesetzt. Dazu wird die Bibliothek Spring-Boot-Starter-Websocket verwendet. Es wird ein WebSocketHandler implementiert, welcher unter dem Pfad "<serverUrl>/signaling" erreichbar ist. Etablierte Verbindungen müssen eindeutig einem Client zugeordnet werden können. Diese Identifikation wird als Query Parameter bei Verbindungsaufbau mitgegeben. Der WebSocketHandler definiert Methoden die beim Öffnen und Schliessen von Verbindungen sowie beim Empfang von Signalen aufgerufen werden. Die Verarbeitung dieser Signale und Verbindungen wird an den SignalingService delegiert.

### 7.4.3 Sicherheit für Signaling

Der Zugriff auf die Signaling Instanz und die darüber ausgetauschten Signale darf nur für Berechtigte möglich sein. Um dies sicherzustellen, wird der Verbindungsaufbau nur erlaubt, wenn die Anfrage dazu authentisiert ist. Für die Authentisierung wird derselbe Mechanismus wie für Http-Anfragen der Cloudservice-API verwendet. Durch die Konfiguration des Cloudservices wird die Authentifizierung aller Http Requests überprüft. Mit dieser Prüfung wird sichergestellt, dass gültige Authentifizierungsdaten im Header der Anfrage vorhanden sind. Ist dies nicht der Fall, wird eine entsprechende Fehlermeldung zurückgegeben. Bei der Prüfung von Http-Anfragen durch den Cloudservice werden weiter die Rollen, welche dem Aufrufer zugewiesen sind ausgelesen.

Diese Prüfung der Authentifizierung wird auch für die Http-Anfragen, welche zum Aufbau einer WebSocketverbindung nötig sind ausgeführt. Beinhaltet eine Anfrage zum Aufbau einer WebSocketverbindung keine gültige Authentifizierung, wird eine Fehlermeldung zurückgegeben. Der Aufbau der WebSocketverbindung wird abgebrochen.

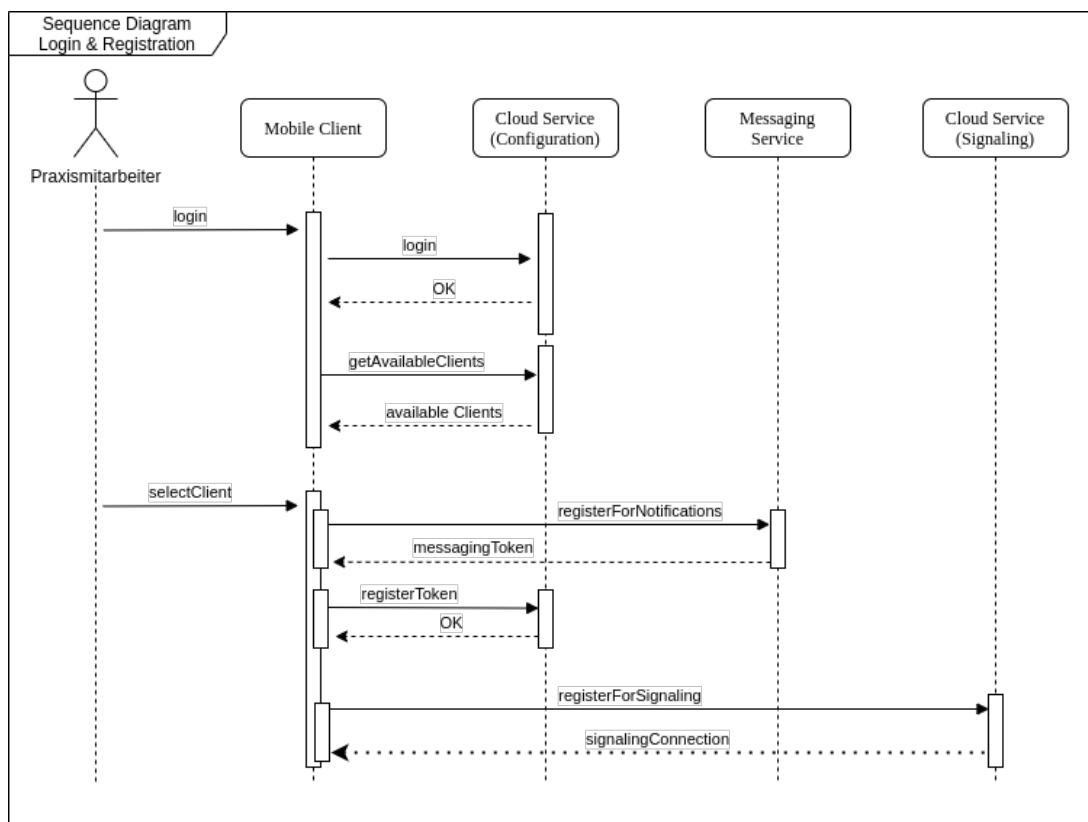
Die Prüfung der ausgelesenen Rollen wird mit der Klasse HttpSessionHandshakeInterceptor implementiert. Diese wird, nachdem eine Anfrage zum Aufbau einer WebSocketverbindung eingegangen ist, aufgerufen. Der HttpSessionHandshakeInterceptor erlaubt es die Anfrage zum Verbindungsaufbau auszulesen. Dies erlaubt es hier zu verifizieren, dass der Request authentifiziert wurde und nur die zugeteilten Rollen zu überprüfen. Praxisruf kennt die zwei Rollen "ADMIN" und "USER". Beide Rollen sind berechtigt, dass Rufsystem über den Mobile Client zu verwenden und dürfen damit Signalmeldungen austauschen. Hat der Aufrufer keine dieser Rollen, wird der Verbindungsaufbau abgebrochen.

Für WebSocketverbindungen wird ausschliesslich das Protokoll Secure WebSockets (WSS) verwendet. Die Http-Anfragen für den Verbindungsaufbau werden ausschliesslich über Https versendet. Der Verbindungsaufbau und Austausch von Signalmeldungen über das Signaling Modul sind damit verschlüsselt.

#### 7.4.4 Anmeldung und Registrierung

Dieses Kapitel beschreibt, welche Anmelde- und Registrierungsprozesse im Mobile Client benötigt werden.

Anmeldung und Registrierung für Benachrichtigungen funktionieren mit dem neuen Mobile Client nach demselben Ablauf wie im Vorgängerprojekt. Die Registrierung für Sprachverbindungen beim Signaling Modul des Cloudservice wird mit diesem Projekt hinzugefügt. Der gesamte Ablauf von Anmeldung und Registrierung wird in Abbildung 7.16 dargestellt. Praxismitarbeitende öffnen die Applikation und geben ihr Benutzername und Passwort ein. Der Mobile Client verwendet diese, um sich über Basic Authentication beim Cloudservice anzumelden. Als Antwort auf die Anmeldung gibt der Cloudservice ein Json Web Token (JWT) zurück. Dieses wird lokal auf dem Gerät gespeichert und für alle weiteren Anfragen an den Cloudservice verwendet. Nachdem die Anmeldung erfolgt ist, wird eine Liste der verfügbaren Konfigurationen geladen. Der Benutzer wählt die gewünschte Konfiguration aus und bestätigt.



**Abbildung 7.16:** Sequenzdiagramm - Anmeldung und Registrierung

Danach wird diese Konfiguration geladen und die Hauptansicht angezeigt. Die geladene Konfiguration beinhaltet alle Informationen die nötig sind um Buttons für Benachrichtigungen und Sprachverbindungen anzuzeigen. Im Hintergrund muss sich der Mobile Client nun für Benachrichtigungen und Sprachverbindungen registrieren. Für Benachrichtigungen registriert er sich zuerst bei Firebase Cloud Messaging. Er erhält ein Token, welches den Client beim Messaging Service identifiziert. Dieses Token sendet der Mobile Client zusammen mit der gewählten Konfiguration an den Cloudservice. Dieser persistiert die Registrierung und kann sie verwenden, um Benachrichtigungen an diesen Client zuzustellen. Für Sprachverbindungen muss zudem eine Verbindung zum Signaling Modul des Cloudservices aufgebaut werden. Dazu wird wie in Kapitel 7.4.6 beschrieben eine WebSocketverbindung geöffnet.

### 7.4.5 Signalmeldungen

Dieses Kapitel beschreibt die Signalmeldungen, welche in Praxisruf verwendet werden. Dabei wird beschrieben wie diese aufgebaut sind und welchen Rolle sie im System erfüllen. Abbildung 7.17 zeigt, wie Signalmeldungen im Mobile Client modelliert werden.

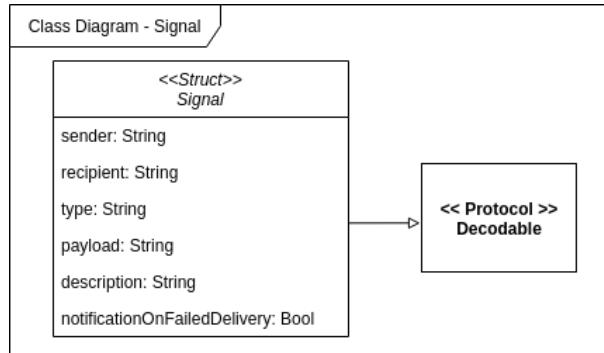


Abbildung 7.17: Klassendiagramm - Signal

Alle Signalmeldungen beinhalten Identifikation von Sender und Empfänger. Diese werden von der Signaling Instanz verwendet, um die Signale korrekt weiterzuleiten. Die Werte Type, Payload und Description werden im Mobile Client verwendet, um das Signal korrekt zu verarbeiten und Verbindungen aufzubauen. Das Flag notificationOnFailedDelivery wird im Cloudservice ausgewertet. Wenn ein Signal nicht zugestellt werden kann und dieses Flag aktiviert ist, wird der Empfänger mit einer Benachrichtigung darüber informiert. Dazu wird die API des Notification Moduls des Cloudservices verwendet.

Praxisruf verwendet sechs Typen von Signalmeldungen. Folgende Übersicht beschreibt pro Typ welche Daten ein Signal beinhaltet und welche Rolle es erfüllt.

Offer	Wird vom Initiator der Sprachverbindung an die Empfänger gesendet. Ein Offer beinhaltet die SDP Informationen des Initiators und initiiert eine Verbindung.
Answer	Wird vom Empfänger eines Offers an den Initiator der Sprachverbindung gesendet. Es beinhaltet SDP Informationen des Empfängers und bestätigt eine Verbindung.
Ice Candidate	Beinhaltet Informationen eines ICE Candidates, die für den Verbindungsaufbau verwendet werden. Nach Verarbeitung von Offer und Answer tauschen Initiator und Empfänger solange Ice Candidate Signale aus, bis sie sich auf einen Kandidaten geeinigt haben. Dieser wird verwendet, um die Verbindung aufzubauen.
End	Wird versendet, nachdem die Verbindung durch tippen des Auflegen-Button in der Applikation beendet wurde. Empfang dieses Signal führt dazu, dass offene Sprachverbindungen zum Sender dieses Signals beim Empfänger beendet werden.
Unavailable	Wenn der Signalingserver ein Signal nicht zustellen kann, wird ein Unavailable-Signal zurück an den Sender gesendet. Der Sender ist so informiert, dass die Verbindung zum Gesprächspartner nicht aufgebaut wurde.
Decline	Wird ein Offer empfangen während die Gegensprechanlage in den lokalen Einstellungen deaktiviert ist oder bereits in Anruf aktiv ist, sendet der Mobile Client ein Decline Signal zurück. Der Sender ist so informiert, dass die Verbindung zum Gesprächspartner abgelehnt wurde.

### 7.4.6 Verbindungsauflaufbau

Praxismitarbeitende können Sprachverbindungen zu anderen Clients aufbauen indem sie auf den entsprechenden Button in der Praxisruf App tippen. Zum Zeitpunkt an dem der Button getippt wird, weiss der Mobile Client noch nicht, zu welchen Clients diese Verbindung aufgebaut werden soll. Als Erstes muss deshalb beim Cloudservice angefragt werden, zu welchen Clients eine Sprachverbindung aufgebaut werden soll. Der Cloudservice bietet dazu einen Endpoint an, über den der vollständige CallType des verwendeten Buttons geladen werden kann. Nachdem diese Informationen geladen sind, können Sprachverbindungen zu allen Empfängern aufgebaut werden. Dazu müssen Offer, Answer und Ice Candidate Signale ausgetauscht werden. Der auslösende Client initialisiert die Peer-to-Peer Verbindung auf seiner Seite und sendet für jeden Gesprächspartner ein Offer. Der Cloudservice findet die Verbindung der jeweiligen Empfänger und leitet die Signale über deren Verbindung weiter.

Nach Eingang des Offer-Signals wird die Verbindung auf Empfängerseite initialisiert und eine Answer zurück an den Initiator gesendet. Diese wird gleich wie das Offer-Signal über die Signaling Instanz zugestellt. Der Initiator empfängt die Antwort Signale und ergänzt die notwendigen Verbindungsinformationen. Abbildung 7.18 visualisiert diesen Ablauf.

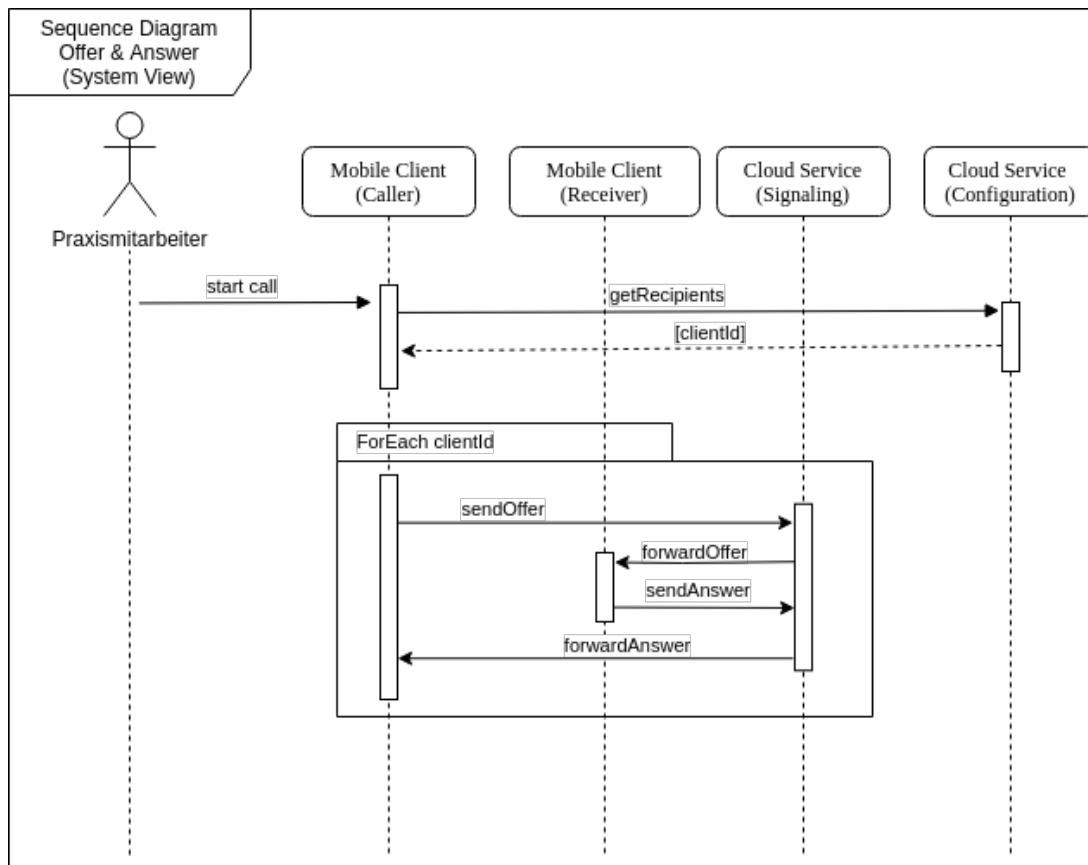


Abbildung 7.18: Sequenzdiagramm - Verbindungsauflaufbau mit Offer und Answer Signalen

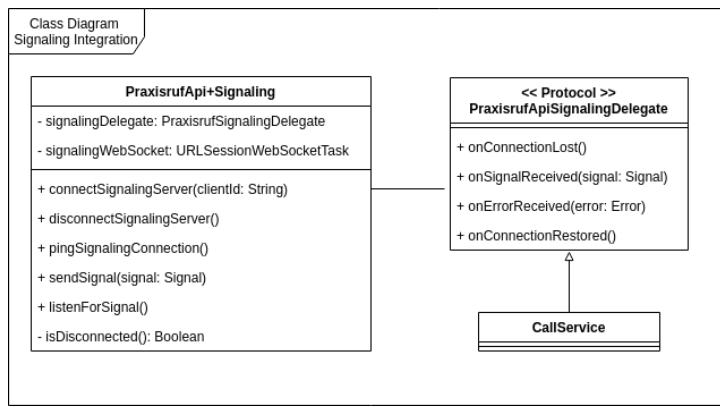
Nachdem die Offer und Answer Meldungen ausgetauscht sind, müssen Ice Candidate Meldungen ausgetauscht werden. Der Austausch von Ice Candidate Meldungen wird so lange wiederholt, bis sich beide Seiten einer Verbindung auf Verbindungsdetails geeinigt haben. Sobald dieser Austausch beendet ist, besteht die Verbindung und es können Sprachdaten ausgetauscht werden.

### 7.4.7 Anbindung Mobile Client an Singaling Instanz

Dieses Kapitel beschreibt, wie Websockeverbindungen zum Austausch von Signalmeldungen in den native Mobile Client integriert werden.

In Kapitel 7.2 wird die Klasse PraxisrufApi beschrieben. Diese implementiert die Anbindung und die Http-API des Cloudservice. Um Signalmeldungen über die Signaling Instanz des Cloudservice auszutauschen, müssen Websockets verwendet werden. Damit die Anbindung an alle Schnittstellen des Cloudservice einheitlich bleibt, wird PraxisrufApi erweitert, um auch WebSocketverbindungen zu unterstützen. Dies beinhaltet den Auf- und Abbau von Verbindungen, sowie das Senden und Empfangen von Meldungen über diese Verbindung. Weiter müssen Verhalten beim Empfang von Fehlermeldungen und dem unerwarteten Schliessen der Verbindung definiert werden.

Der Austausch von Signalmeldungen ist der einzige Anwendungsfall für Websockets in Praxisruf. Deshalb wird auf eine generische Integration von Websockets verzichtet. Zur Erweiterung von PraxisrufApi werden die Extension PraxisrufApi+Signaling und das Protokoll PraxisrufApiSignalingDelegate definiert. Diese setzen die Anbindung von Websockets für den Austausch von Signalmeldungen um.



**Abbildung 7.19:** Klassendiagramm - Signaling Schnittstelle in Mobile Client

Die Extension PraxisrufApi+Signaling ist für die Verbindung zu der Signaling Instanz verantwortlich. Für die Integration dieser Extension in den Rest der Applikation wird das Protokoll PraxisrufApiSignalingDelegate definiert. Zum Aufbau der Verbindung zu der Signaling Instanz wird die Methode connectSignalingServer definiert. Die Identifikation des Clients wird bei dieser Abfrage als Parameter mitgegeben. So kann die Verbindung von der Signaling Instanz eindeutig einem Client zugeordnet werden. Nachdem die Verbindung geöffnet ist, können Signale empfangen und verarbeitet werden. Dies wird durch die Methode listenForSignal initialisiert. Darin wird der WebSocketverbindung signalisiert, dass der Client bereit ist die nächste Meldung zu empfangen. Sobald eine Meldung empfangen wird, wird diese über den PraxisrufApiSignalingDelegate verarbeitet. Wurde eine gültige Signalmeldung empfangen, wird diese über die Methode onSignalReceived verarbeitet. Wurde hingegen eine ungültige Signalmeldung oder eine Fehlermeldung empfangen wird die Methode onErrorReceived aufgerufen. Im Fehlerfall wird zudem überprüft, ob die Verbindung noch offen verwendbar ist. Sollte die Verbindung nicht mehr verwendbar sein, wird die Methode onConnectionLost des Delegates aufgerufen. Dadurch wird versucht, die Verbindung erneut aufzubauen. Ist dies nicht möglich, wird eine Fehlermeldung angezeigt.

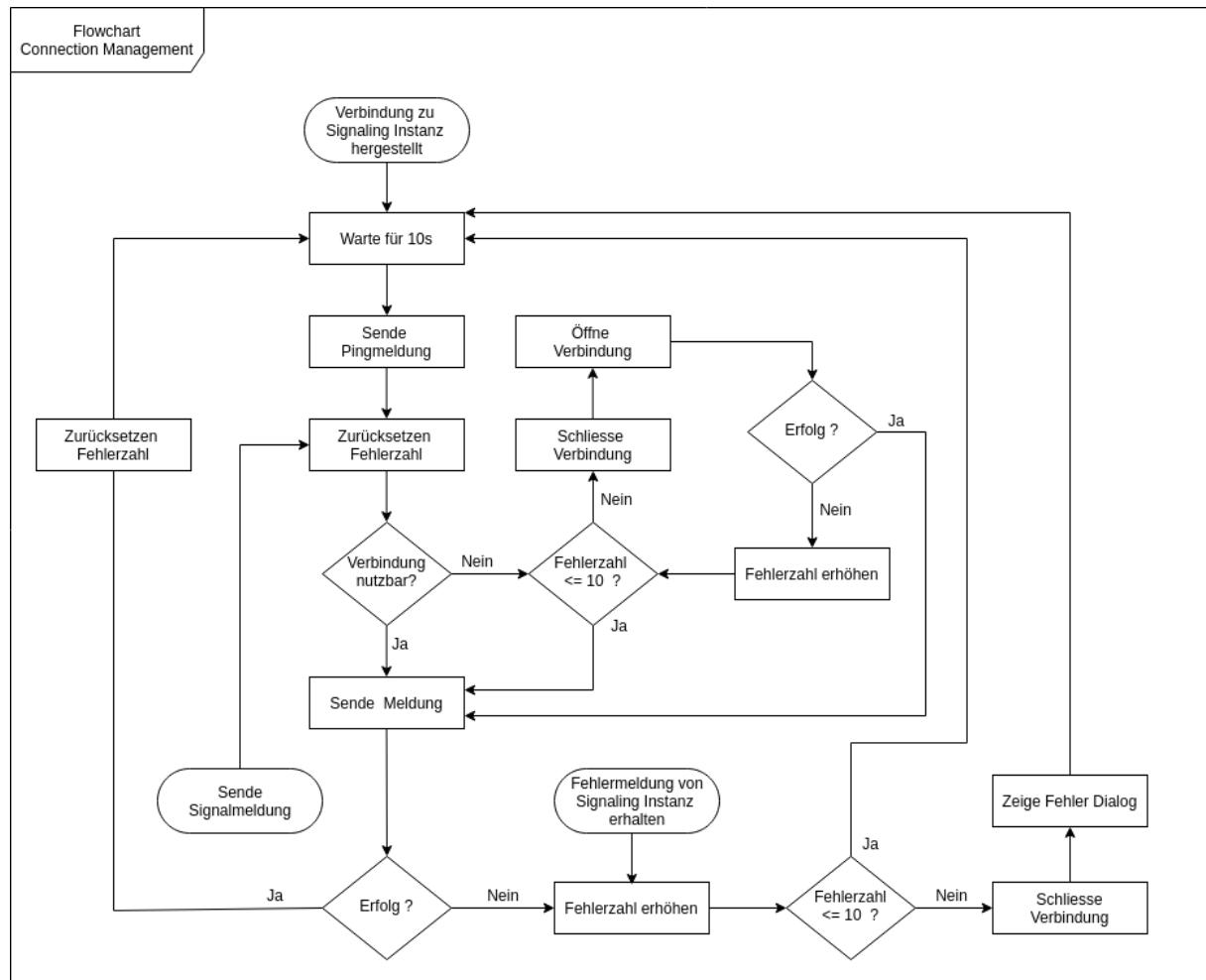
PraxisrufApi+Signaling definiert weiter Methoden um Signal- und Pingmeldungen zu versenden. Pingmeldungen werden in Regelmässigen Abständen gesendet um sicherzustellen, dass die Verbindung zu der Signaling Instanz geöffnet bleibt. Vor dem Senden jeder Meldung wird geprüft, ob die Verbindung zur Signaling Instanz verwendbar ist. Ist dies nicht der Fall, wird die Methode onConnectionLost des

Delegates aufgerufen und anschliessend versucht die Meldung zu versenden. Schlägt das Senden fehl, wird die Methode `onErrorReceived` auf dem Delegate aufgerufen.

Die Methoden des Protokolls CallClientDelegate werden durch die Klasse CallService implementiert. Der CallService vermittelt Anfragen zwischen Signaling Instanz, Benutzeroberfläche und Peer-To-Peer Verbindungen im Mobile Client. Er ist dafür verantwortlich Signale an die Peer-To-Peer Verbindung weiterzuleiten, geschlossene Verbindungen wieder zu öffnen und Fehlermeldungen anzuzeigen. Empfangene Signalmeldungen werden über die Methode onSignalReceived an die Komponente CallClient übergeben (Siehe Kapitel 7.4.9). Die Methoden onConnectionLost und onErrorReceived werden verwendet, um Fehlermeldungen anzuzeigen und geschlossene Verbindungen erneut zu öffnen. Nachdem die Verbindung zur Signaling Instanz verloren gegangen ist, wird bis zu zehnmal versucht, die Verbindung erneut zu öffnen. Wenn die Verbindung dadurch nicht repariert werden kann, wird sie geschlossen und dem Benutzer eine Fehlermeldung angezeigt.

## 7.4.8 Verbindungsverwaltung

Dieses Kapitel beschreibt, wie im Mobile Client sichergestellt wird, dass die Verbindung zu der Signaling Instanz langfristig zur Verfügung steht. Abbildung 7.20 gibt einen Überblick, über die Abläufe dazu umgesetzt werden. Dieser Ablauf ist mit den Komponenten CallService und PraxisrufApi+Signaling die, in Kapitel 7.4.7 beschrieben sind, implementiert.



**Abbildung 7.20:** Flowchart - Verbindungsverwaltung

Um festzustellen, ob die eine getrennte Verbindung repariert werden kann, wird im CallService ein Fehlerzähler geführt. Dieser wird beim Start der Applikation mit null initialisiert und bei jedem Verbindungsfehler um eins inkrementiert. Ein Verbindungsfehler tritt auf, wenn eine Fehlermeldung über die Signaling Verbindung empfangen wird und wenn das Senden einer Meldung oder das Wiederherstellen der Verbindung fehlschlägt.

Die Verbindung zur Signaling Instanz wird hergestellt, sobald die Applikation gestartet wird. Nachdem die Verbindung hergestellt wurde, werden im Abstand von zehn Sekunden Pingmeldungen über die Verbindung gesendet. Dadurch wird sichergestellt, dass die Verbindung geöffnet bleibt. Vor dem Versenden jeder Meldung wird der Fehlerzähler zurückgesetzt und es wird überprüft, ob die Verbindung verwendet werden kann. Ist die Verbindung fehlerhaft, werden bis zu zehn Versuche unternommen, sie wiederherzustellen. Anschliessend wird versucht, die Meldung zu versenden. Wenn die Meldung erfolgreich versendet wurde, wird der Fehlerzähler zurückgesetzt und zehn Sekunden gewartet, bis die nächste Pingmeldung versendet wird. Andernfalls wird die Fehlerzahl um eins erhöht. Ist die Fehlerzahl danach grösser als zehn, wird die Verbindung getrennt. In der Benutzeroberfläche wird ein Fehlerdialog angezeigt. Die Verbindung bleibt geschlossen bis die nächste Meldung versendet wird. Dies der Fall, wenn das Interval für die nächste Pingmeldung abgelaufen ist oder wenn ein Anruf über die Benutzeroberfläche gestartet wird.

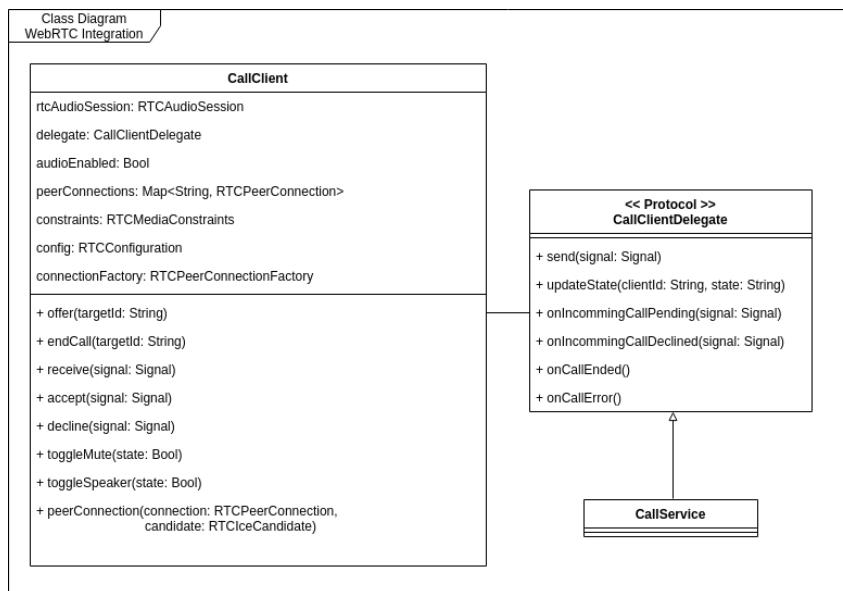
Bei Start eines Anrufes aus der Benutzeroberfläche werden dieselben Prüfungen, wie vor dem Versenden von Pingmeldungen ausgeführt. So werden geschlossene Verbindungen zur Signaling Instanz frühzeitig repariert. Weiter wird nach Empfang jedes Fehlers über die Signaling Verbindung geprüft, ob diese noch offen ist. Ist dies nicht der Fall, werden auch hier bis zu zehn Versuche unternommen, die Verbindung wiederherzustellen.

Dieser Ablauf stellt sicher, dass die Verbindung zu der Signaling Instanz geöffnet bleibt und wenn nötig repariert wird. Wenn möglich, geschieht dies direkt nach Verbindungsverlust. Andernfalls werden Praxismitarbeitende informiert, dass die Verbindung nicht hergestellt werden konnte. Im Hintergrund wird in dieser Situation regelmässig versucht, die Verbindung zu reparieren. Die Bedienelemente für die Gegensprechanlage bleiben während dieser Zeit aktiviert und können weiter verwendet werden. Wenn Praxismitarbeitende einen Anruf starten und die Verbindung immer noch getrennt ist, wird die sie frühzeitig wiederhergestellt. Ist dies erfolgreich, wird der Anruf gestartet. Andernfalls wird erneut ein Fehlerdialog angezeigt.

### 7.4.9 Sprachverbindungen im Mobile Client

Dieses Kapitel beschreibt wie WebRTC-Komponenten in den Mobile Client integriert werden, um Sprachverbindungen zu ermöglichen.

Für die Integration der WebRTC-Komponenten wird eine Klasse CallClient und ein Protokoll CallClientDelegate erstellt. Der CallClient verwaltet Sprachverbindungen, während der Delegate zur Integration mit der restlichen Applikation dient. Abbildung 7.21 zeigt das Klassendiagramm beider Komponenten. Die Methoden des CallClientDelegate-Protokolls werden von der Klasse CallService implementiert. Da diese Klasse auch das Protokoll SignalingDelegate implementiert, kann sie verwendet werden um Signalmeldungen zwischen CallClient und PraxusrufApi+Signaling zu vermitteln.



**Abbildung 7.21:** Klassendiagramm - CallClient und CallClientDelegate

Für den Aufbau von ausgehenden Verbindungen wird eine RTCPeerConnection im CallClient des Senders initialisiert. Es werden die SDP Informationen für Beschreibung der Verbindung erstellt und mit einem Offer-Signal an den Empfänger gesendet. Dieses Offer-Signal wird über den CallClientDelegate versendet, welcher das Signal an die Signaling Instanz zustellt.

Für den Aufbau von eingehenden Verbindungen wird das Offer-Signal über die Methode receive empfangen. Dabei wird die Verbindung nicht direkt initialisiert. Stattdessen wird das Signal zwischengespeichert und die Methode onIncommingCallPending des Delegates aufgerufen. Die Implementation des CallClientDelegate navigiert darauf zu der Ansicht für aktive Anrufe und meldet dem CallClient über die Methode acceptPending, dass die Verbindung initialisiert werden soll. Der CallClient initialisiert darauf eine RTCPeerConnection und sender ein Answer Signal über die send Methode.

Answer-Signale werden ebenfalls über die receive Methode im CallClient empfangen. Beim Empfang einer Answer werden die SDP Informationen auf der lokalen RTCPeerConnection ergänzt.

Neben dem Austausch von Offer- und Answer-Signalen müssen Ice Candidate Signale ausgetauscht werden können. Sobald ein Ice Candidate zur Verfügung steht, erstellt der CallClient ein entsprechendes Signal und sendet es an den Empfänger. Damit dies möglich ist, muss der CallClient über verfügbare Ice Candidates informiert werden. Die WebRTC Bibliothek bietet dazu das Protokoll RTCPeerConnection-Delegate. Dieses kann auf der RTCPeerConnection registriert werden und wird aufgerufen, sobald ein neuer Ice Candidate verfügbar ist. Der CallClient implementiert dieses Protokoll und registriert sich auf allen RTCPeerConnections als Delegate.

Sprachverbindungen müssen über die Benutzeroberfläche verwaltet werden können. Der CallClient bietet deshalb Methoden an, um eine Verbindung zu starten und beenden. Weiter bietet er die Möglichkeit Mikrofon und Lautsprecher für geöffnete Verbindungen stummzuschalten. Bei eingehenden Verbindungen, dem Beenden von Anrufen und Veränderungen am Status einer Verbindung, muss die Benutzeroberfläche informiert werden. Der CallClientDelegate definiert dazu Methoden über welche der CallClient diese Informationen weitergeben kann.

#### 7.4.10 Signalverarbeitung im Mobile Client

Peer-To-Peer Sprachverbindungen werden über die Komponenten CalService, CallClient und PraxisrufApi+Signaling in den Mobile Client integriert. Dieses Kapitel beschreibt den Kommunikationsablauf zwischen diesen Komponenten anhand des Austauschs von Offer- und Answer-Signalen.

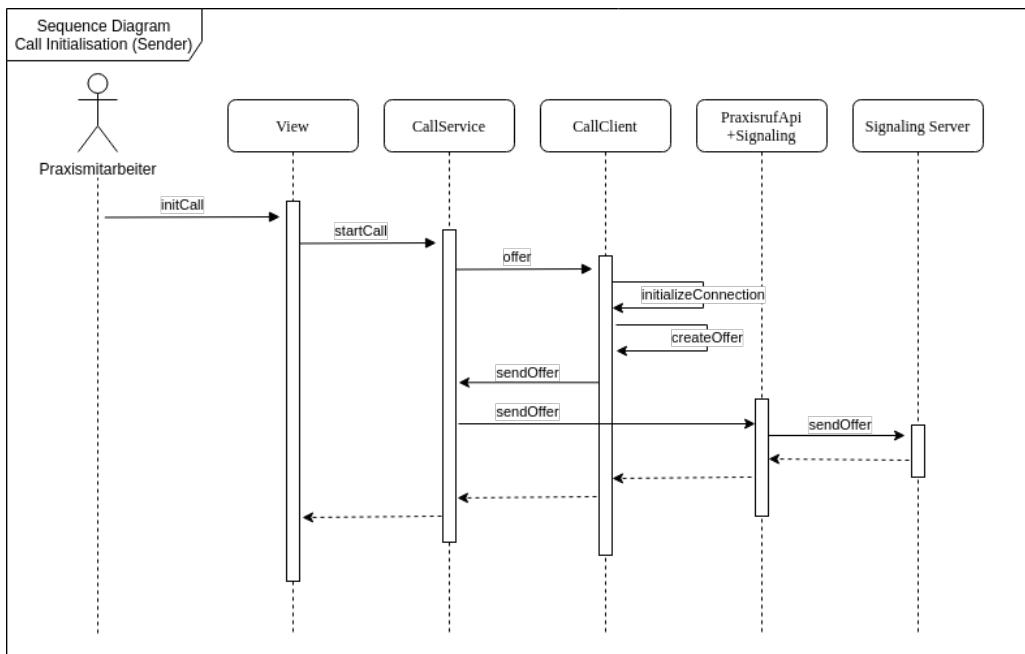
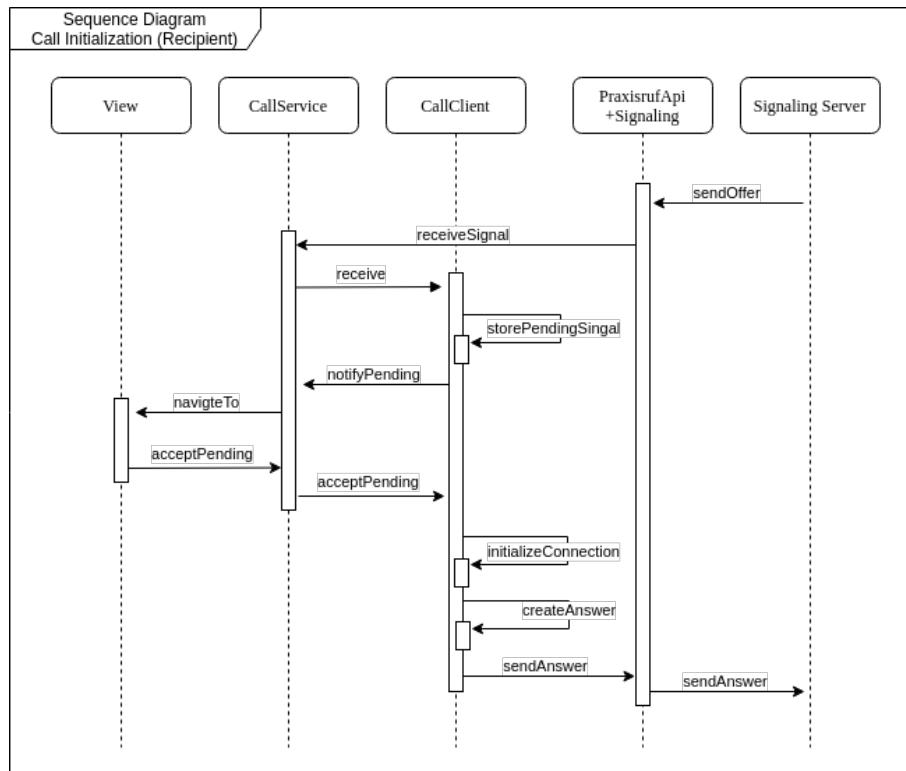


Abbildung 7.22: Sequenzdiagramm - Initialisierung Sprachverbindung auf Senderseite

Die Klassen CallClient und PraxisrufApi+Signaling definieren je ein Delegate-Protokoll. Diese Protokolle definieren die Funktionen, über welche die Komponenten Informationen weitergeben können. Die Klasse CallService implementiert beide Delegate-Protokolle. Er erstellt Instanzen von CallClient und PraxisrufApi+Signaling und registriert sich anschliessend bei beiden als Delegate. Der CallService selbst wird in den View Komponenten der Applikation verwendet. Er nimmt Benutzereingaben entgegen und delegiert die entsprechende Funktionalität an den CallClient und PraxisrufApi+Signaling. Er nimmt ausserdem Informationen von CallClient und PraxisrufApi+Signaling entgegen und stellt Anzeigeeinrichtungen für die Benutzeroberfläche zur Verfügung.

Abbildung 7.22 zeigt die Kommunikation zwischen den beteiligten Komponenten bei der Initialisierung einer Sprachverbindung auf Empfängerseite. Dabei wird der Ablauf von Benutzerinteraktion des Senders bis zum Senden des Offer-Signals dargestellt. Sobald Praxismitarbeitende einen Anruf startet, wird die View für aktive Anrufe geladen. Diese initialisiert den Anruf über den CallService. Der CallService ruft dazu als Erstes den CallClient auf. Der CallClient initialisiert die lokalen Verbindungsinformationen und erstellt ein Signal, um den Empfänger zu informieren. Dieses Signal gibt er an den CallService weiter. Der CallService leitet das Signal an den PraxisrufApi+Signaling weiter, welcher das Versenden an den Cloudservice übernimmt.



**Abbildung 7.23:** Sequenzdiagramm - Initialisierung Sprachverbindung auf Empfängerseite

Abbildung 7.23 zeigt den Ablauf bei Eingang eines Answer-Signals auf Empfängerseite. Das versendete Signal wird über das Signaling Modul des Cloudservice an den Empfänger übermittelt. Dieser empfängt das Signal über die Komponente PraxisrufApi+Signaling. Diese gibt das Signal über die Methode on-SignalReceived an den CallService weiter. Der CallService aktiviert die Ansicht für aktive Anrufe und leitet das Signal an den CallClient weiter. Der CallClient initialisiert die Peer-To-Peer Verbindung und erstellt ein Answer-Signal zur Bestätigung. Dieses Signal wird wiederum über den CallService zum PraxisrufApi+Signaling weiter zum Cloudservice versendet.

#### 7.4.11 Netzwerkvoraussetzungen

Praxisruf ist darauf ausgelegt innerhalb von Arztpraxen verwendet zu werden. Dabei wird pro Zimmer ein Gerät installiert, welches als Endpunkt für das System dient [1]. Damit werden alle Endgeräte innerhalb derselben Praxis betrieben. Dies erlaubt es, die Geräte über ein lokales Netzwerk zu verbinden. Ist diese Voraussetzung gegeben, kann der Verbindungsaufbau vereinfacht werden. Da die Geräte direkt im lokalen Netzwerk kommunizieren können, ist es nicht nötig, die Protokolle STUN oder TURN zu verwenden. Damit wird es möglich auf den Betrieb eines ICE Servers zu verzichten. Die Geräte können ICE Kandidaten für direkte Kommunikation im lokalen Netzwerk austauschen.

Die Integration von WebRTC in der iOS App wird unter der Voraussetzung umgesetzt, dass alle beteiligten Geräte im selben Netzwerk betrieben werden. Dies vereinfacht den Aufbau von Verbindungen und erlaubt es, neben der Signaling Instanz keine weitere Infrastruktur betreiben zu müssen. Sprachverbindungen mit Praxisruf sind deshalb nur möglich, wenn die beteiligten Geräte im selben lokalen Netzwerk betrieben werden. Das Versenden und Empfangen von Benachrichtigungen funktioniert wie im Vorgängerprojekt auch ausserhalb des lokalen Netzwerkes. Damit können auch für Empfänger, die nicht im lokalen Netzwerk sind, über verpasste Anrufe mit Benachrichtigungen informiert werden.

## 8 Umsetzung

Dieses Kapitel beschreibt und evaluiert die erzielten Ergebnisse dieser Arbeit. Zunächst wird eine Übersicht zu den erreichten Zielen und Resultaten gegeben. Anschliessend wird die Funktionsweise des Systems anhand von Screenshots des neuen Mobile Clients beschrieben. Weiter werden die Resultate der durchgeführten Funktions- und Performancetests beschrieben. Dabei wird auch darauf eingegangen, in welchen Bereichen das umgesetzte System weiter verbessert werden kann. Zum Schluss des Kapitels wird ein Fazit gezogen.

### 8.1 Resultate

Mit dieser Arbeit wurde das cloudbasierte Praxisrufsystem "Praxisruf" erweitert. Das erweiterte Rufsystem erlaubt es, Benachrichtigungen zu Versenden und über Sprachverbindungen zu kommunizieren. Es wurde eine native iOS App entwickelt, welche es erlaubt Praxisruf zu bedienen. Diese App ersetzt den Mobile Client aus dem Vorgängerprojekt und unterstützt dabei alle Funktionen des alten Clients. Die Systemarchitektur sowie die Komponenten Cloudservice und Admin UI wurden erweitert, um die neuen Funktionen ermöglichen.

Mit Amazon Polly wurde ein Service für Sprachsynthese an das System angebunden. Diese Anbindung wird verwendet, um den Inhalt empfangener Benachrichtigungen automatisch vorzulesen. Letztlich wurde WebRTC verwendet, um eine konfigurierbare Gegensprechanlage zu implementieren. Diese erlaubt es mit der iOS App Sprachverbindungen über das Praxisrufsystem aufzubauen. Sowohl das Vorlesen von Benachrichtigungen, als auch die Gegensprechanlage sind über eine Weboberfläche konfigurierbar.

Die zu Projektbeginn definierten Meilensteine M01 bis M08 wurden erreicht. Die Funktionalen Anforderungen an das System wurden umgesetzt. Die bestehende Infrastruktur aus dem Vorgängerprojekt wurde für dieses Projekt übernommen. Umgesetzt wurden die Meilensteine mit folgenden User Stories:

- U01 - Migration bestehender Funktion im Mobile Client
- U02 - Benachrichtigungen vorlesen
- U03 - Vorlesen von Benachrichtigungen deaktivieren
- U04 - Button für Sprachverbindung 1:1
- U05 - Button für Sprachverbindung 1:n
- U06 - Über Sprachverbindung in Echtzeit kommunizieren
- U07 - Nur relevante Buttons für Sprachverbindungen anzeigen
- U08 - Benachrichtigungston für eingehende Sprachverbindungen
- U09 - Empfangene und verpasste Sprachverbindungen in Inbox anzeigen
- U10 - Eingehende Sprachverbindungen automatisch öffnen
- U11 - Sprachverbindung beenden
- U13 - Vorlesen von Benachrichtigungen konfigurieren
- U14 - Buttons für Sprachverbindungen konfigurieren
- U15 - Konfiguration über Admin UI vornehmen
- U16 - Bestehende Infrastruktur übernehmen
- U17 - Weiterverwendung bestehender Komponenten
- U18 - Native iOS Applikation
- U19 - Für Externe Services Amazon Webservices verwenden

Der zu Beginn definierte Projektplan konnte grösstenteils eingehalten werden. Setup- und Konzept-Phase wurden im geplanten Zeitraum abgeschlossen. Während der Umsetzung ist es hingegen zu mehreren Verzögerungen gekommen. Die Umsetzung und Testen der Gegensprechanlage hat deutlich mehr Zeit beansprucht, als eingeplant wurde. Insbesondere die Integration und effiziente Verwendung von WebRTC

im nativen iOS Client war anspruchsvoller als erwartet. Dies ist einerseits auf mangelhafte Dokumentation der verwendeten Bibliotheken zurückzuführen. Andererseits wurde schlicht zu wenig Puffer für die Implementation und unerwartete Probleme eingeplant. Die Verzögerungen während der Umsetzung hat dazu geführt, dass weniger Zeit als geplant für Polishing und Erweiterungen aufgewendet werden konnten. Dementsprechend konnte der Meilenstein M09 nicht vollständig erreicht werden. Das entwickelte System hat in den Bereichen Stabilität und Benutzerfreundlichkeit noch Lücken.

Der Meilenstein M10 betrifft Abschluss und Abgabe des Projektberichts und Entwicklung des Systems. Er ist mit dem Abschluss dieses Projektes erfüllt.

### 8.1.1 Native iOS Applikation

Dieses Kapitel beschreibt das umgesetzte Praxisrufsystem anhand der Ansichten der iOS Applikation. Sämtliche in diesem Kapitel dargestellten Ansichten wurden als Screenshots auf einem iPad 9. Generation erstellt.

#### Anmeldung und Konfiguration

Der Mobile Client bietet eine einfaches Verfahren zur Anmeldung und Konfiguration. In einem ersten Schritt gibt der Praxismitarbeitende in der Login-Ansicht Benutzername und Passwort ein (Abbildung 8.1). Anschliessend kann er auf einer zweiten Ansicht, die gewünschte Zimmerkonfiguration wählen (Abbildung 8.2). Benutzer, Passwort und die gewählte Konfiguration werden dabei auf dem Gerät gespeichert. Bis sich der Benutzer manuell abmeldet, erfolgt die Anmeldung bei allen zukünftigen Starts der App automatisch. Dabei wird die gespeicherte Kombination von Benutzer und Konfiguration wieder verwendet.

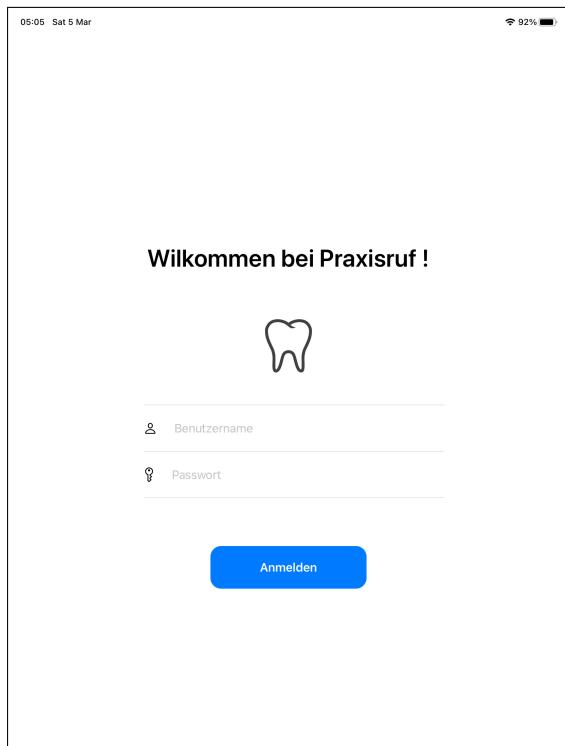


Abbildung 8.1: Ansicht Login

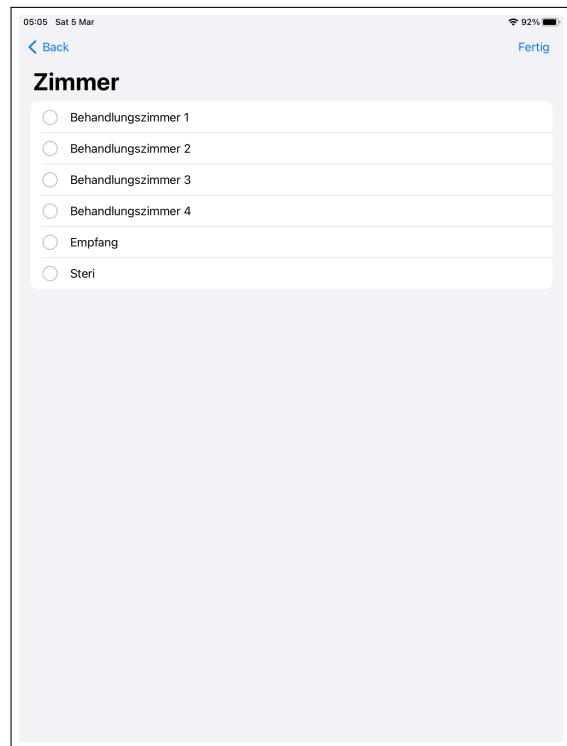
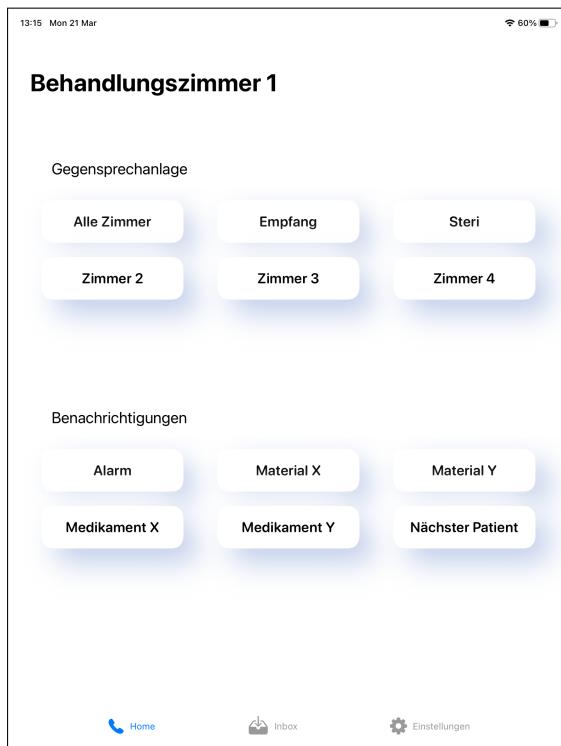


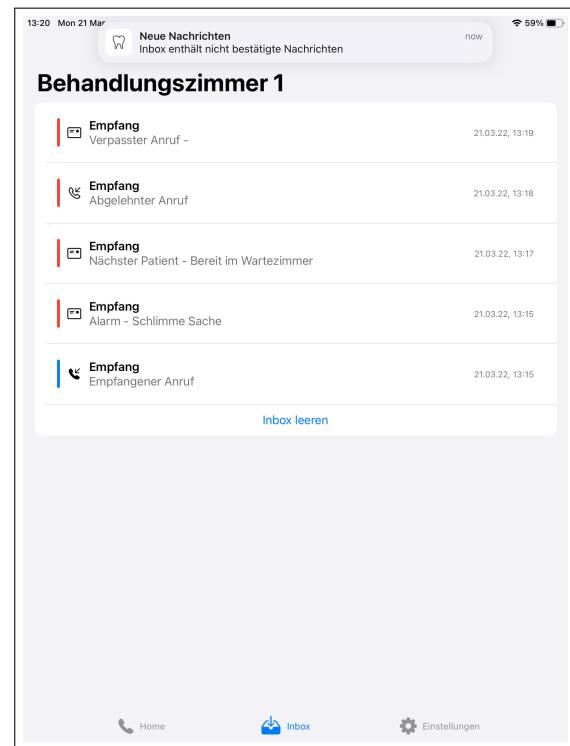
Abbildung 8.2: Ansicht Zimmerwahl

## Startseite und Inbox

Nach der Anmeldung und Konfigurationsauswahl wird der Benutzer auf die Hauptansicht der App weitergeleitet. Über eine Navigationsleiste am unteren Bildschirmrand kann zwischen den Bereichen Home, Inbox und Einstellungen navigiert werden. Der Bereich Home (Abbildung 8.3) ist in zwei Teile gegliedert. Diese beinhalten Buttons, über welche Benachrichtigungen versendet und Sprachverbindungen gestartet werden können. Welche Buttons zur Verfügung stehen werden durch die gewählte Zimmerkonfiguration vorgegeben und wurden im Vorfeld von Praxisadministrierenden konfiguriert. Der Bereich Inbox (Abbildung 8.4) zeigt eine Liste von empfangenen Benachrichtigungen sowie verpassten und vergangenen Anrufen. Einzelne Einträge in dieser Liste können durch eine Wischgeste (Swipe left) quittiert und damit entfernt werden. Weiter können mit der Schaltfläche "Inbox leeren" am unteren Ende der Liste alle Einträge gleichzeitig quittiert werden. Das quittieren von Inboxelementen geschieht ausschliesslich lokal auf dem Gerät des Empfängers. Der Sender von Benachrichtigungen wird nicht über die Quittierung informiert. Damit wurde das Quittieren von Elementen analog zum Vorgängerprojekt umgesetzt.



**Abbildung 8.3:** Ansicht Home



**Abbildung 8.4:** Ansicht Inbox

Ein Hintergrundprozess prüft in regelmässigen Abständen, ob es unquittierte Elemente in der Inbox gibt. Ist dies der Fall wird eine Push-Benachrichtigung angezeigt und der Benachrichtigungston abgespielt. Diese Benachrichtigung ist in Abbildung 8.4 zu sehen. Eine Ausnahme bilden Elemente für eingehende Anrufe, die erfolgreich empfangen wurden. Diese werden in der Inbox angezeigt und können quittiert werden. Sie werden aber bei der Prüfung auf nicht quittierte Elemente ignoriert.

## Einstellungen und Aktive Anrufe

Im Bereich Einstellungen (Abbildung 8.5) werden Informationen zur gewählten Zimmerkonfiguration und dem angemeldeten Benutzer angezeigt. Weiter können lokale Einstellungen vorgenommen werden. Das Vorlesen von empfangenen Benachrichtigungen sowie das Empfangen von Anrufen kann hier deaktiviert werden. Ist das Vorlesen von Benachrichtigungen deaktiviert, wird der Inhalt von Benachrichtigungen nie vorgelesen. Benachrichtigungen können aber weiterhin empfangen und versendet werden. Ist der Empfang von Anrufen deaktiviert, werden alle eingehenden Anrufe automatisch abgelehnt. Dem Empfänger wird dabei eine Benachrichtigung angezeigt, die über den verpassten Anruf informiert. Der Anrufer wird über eine Signalmeldung informiert, dass die Verbindung nicht hergestellt werden konnte. Der Status des Empfängers wird in der Ansicht für Aktive Anrufe auf Senderseite entsprechend dargestellt. Über einen Button kann der Benutzer sich zudem von der App abmelden.

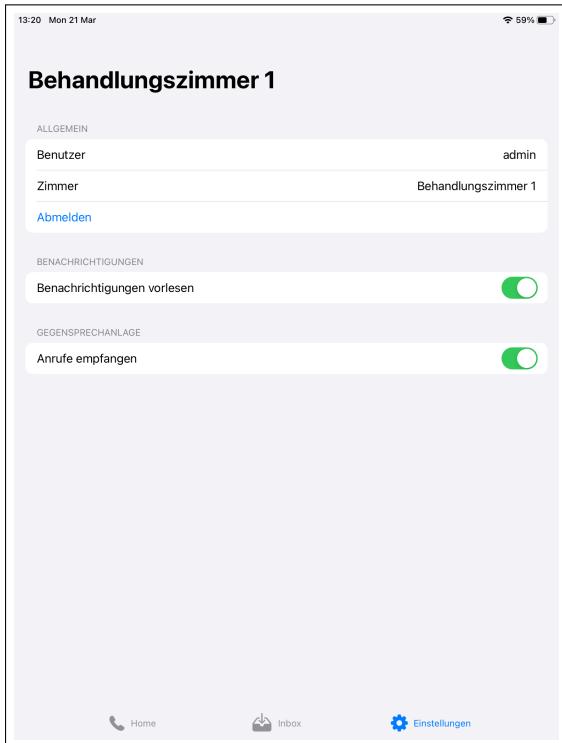


Abbildung 8.5: Ansicht Einstellungen

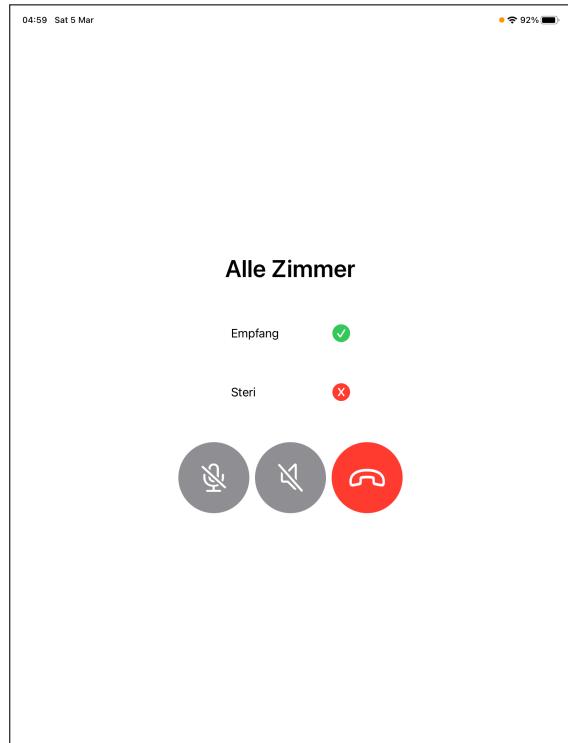


Abbildung 8.6: Ansicht Aktiver Anruf

Die Ansicht "Aktiver Anruf" (Abbildung 8.6) wird angezeigt, nachdem ein Anruf gestartet wurde. Entweder durch antippen eines Buttons in der Home Ansicht oder beim Empfang eines Anrufes von einem anderen Endgerät. In dieser Ansicht wird der Titel des gestarteten Anrufes bzw. Name des Zimmer des Gesprächspartners angezeigt. Wenn mehr als ein Gesprächspartner am Anruf beteiligt ist, wird zudem eine Liste der Teilnehmer zusammen mit deren Verbindungsstatus angezeigt. Die Statusanzeige unterscheidet zwischen "Verbindung erfolgreich", "Verbindung nicht erfolgreich" und "Verbindung in Arbeit". Allen Gesprächsteilnehmern stehen Buttons zur Stummschaltung des eigenen Lautsprechers und Mikrofons zur Verfügung. Zudem können alle Gesprächsteilnehmenden die Unterhaltung durch den roten Auflegen Button beenden.

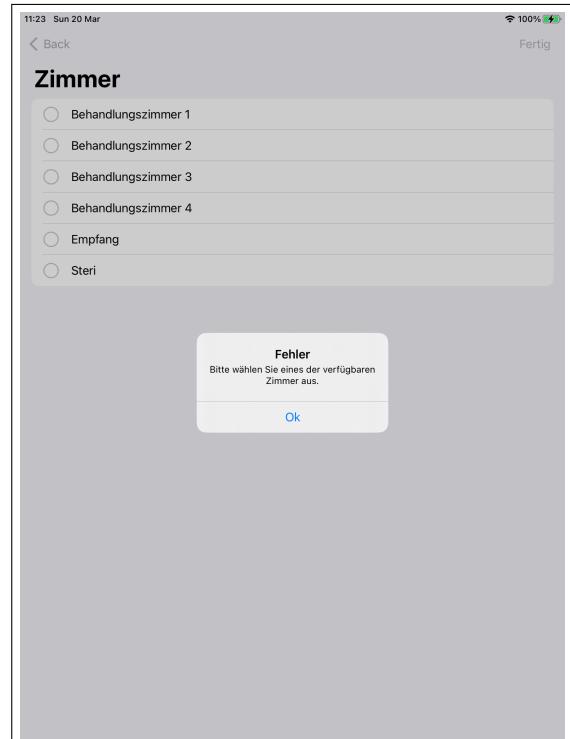
In Abbildung 8.6 wird ein Aktiver Anruf mit den Empfängern Empfang und Steri dargestellt. Dabei hat die Verbindung zu Empfang den Status "Verbindung hergestellt" und die Verbindung zu Steri den Status "Verbindung nicht erfolgreich". Konnte die Verbindung zu einem Empfänger nicht hergestellt werden, wird dieser mit einer Benachrichtigung darüber informiert.

## Hintergrundbenachrichtigungen und Fehlerhandling

Benachrichtigungen können mit Praxisruf auch empfangen werden, wenn die Praxisruf-App nicht aktiv ist. Im Hintergrund empfangene Benachrichtigungen erscheinen als Push Benachrichtigungen auf dem Home Screen des iPads. Anrufe über die Gegensprechsanlage können nur empfangen werden, wenn die Applikation geöffnet ist. Ist die App minimiert oder beendet, ist der jeweilige Client für Gespräche nicht verfügbar. Ein nicht verfügbarer Client wird über Hintergrundbenachrichtigungen auf verpasste Anrufe hingewiesen. Abbildung 8.7 zeigt eine Benachrichtigung "Alarm" und eine Benachrichtigung für einen Verpassten Anruf aus dem Zimmer Empfang.



**Abbildung 8.7:** Hintergrund Benachrichtigung



**Abbildung 8.8:** Ansicht Zimmerwahl - Fehlerdialog

Fehler in der Applikation werden dem Benutzer in einem einfachen Dialogfenster angezeigt. Abbildung 8.8 zeigt eine Fehlermeldung, die bei der Auswahl der Zimmerkonfiguration auftreten kann. Wenn die Wahl der Konfiguration bestätigt wird, ohne dass eine Konfiguration ausgewählt wurde, wird eine entsprechende Fehlermeldung angezeigt.

## 8.2 Tests

### 8.2.1 Benutzertests

Am 11.01.2022 wurden zusammen mit dem Auftraggeber Benutzertests durchgeführt. Die iOS Applikation wurde auf zwei Physische iPads installiert. Eine dritte Instanz der Applikation wurde auf einem Simulator gestartet. Während den Benutzertests wurde folgendes getestet:

1. Anmeldung und Konfigurationsauswahl funktioniert wie im Vorgängerprojekt.
2. Benachrichtigungen können zwischen Mobile Clients versendet werden.
3. Empfangene Benachrichtigungen werden dem Benutzer vorgelesen.
4. Sprachverbindungen können über Buttons hergestellt werden.
5. Sprachverbindungen werden automatisch angenommen.
6. Über Sprachverbindungen können in Echtzeit Unterhaltungen geführt werden.
7. Sprachverbindungen können wieder getrennt werden.

Der Kunde hat während den Tests folgende Verbesserungswünsche eingebracht:

1. Das Quittieren von Einträgen in der Inbox soll diese direkt löschen.
2. Das Quittieren von Einträgen in der Inbox soll durch eine Wischgeste möglich sein.
3. Die Töne für Benachrichtigungen und eingehende Anrufe von den Tönen anderer Apps unterschieden werden können.
4. Die Töne für Benachrichtigungen und eingehende Anrufe sollen konfigurierbar sein.
5. Icon und Name der App sollen angepasst werden.
6. Bei eingehenden Sprachverbindungen soll ein Benachrichtigungston erklingen.
7. Status von Verbindungsteilnehmern soll als Icon dargestellt werden.

Dieses Feedback konnte grösstenteils umgesetzt werden. Elemente können per Wischgeste quittiert werden und werden dabei direkt aus der Inbox gelöscht. Icon und Name der App wurden angepasst.

Bei eingehenden Sprachverbindungen ertönt ein Benachrichtigungston bevor die Verbindung geöffnet wird. Nachdem die Verbindung geöffnet wird, wird der Status aller Teilnehmenden mit einem einfachen Icon dargestellt. Es werden unterschiedliche Töne für Benachrichtigungen, Erinnerungen an unquittierte Nachrichten und den Eingang von Sprachverbindungen verwendet. Dabei werden keine Systemtöne von iOS verwendet. Dadurch können Praxismitarbeitende die Bedeutung eines Tons aus der Praxisruf-App eindeutig erkennen. Damit sind die Anforderungen aus Punkten 1. bis 3. sowie 5. bis 7. erfüllt.

Der Punkt 4. konnte aus Zeitgründen nicht umgesetzt werden. Alle Töne sind allerdings fest definiert und können nicht durch den Benutzer konfiguriert werden. Eine Änderung der Töne ist nur über Änderungen am Quellcode möglich.

## 8.2.2 Funktionstests

Im Rahmen des Projektes Peer-To-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem wurde ein finaler Testplan definiert. Diese Tests wurden zum Abschluss des Projektes durchgeführt, um die Funktionalität des Systems abschliessend zu testen. Die Szenarien S01 bis S18 wurden dabei aus dem Vorgängerprojekt "IP5 Cloudbasiertes Praxisrufsystem" übernommen. Dadurch kann sichergestellt werden, dass alle Funktionen aus dem Vorgängerprojekt korrekt migriert wurden. Die Testszenarien S19 bis S40 behandeln die Funktionen Sprachsynthese und Gegensprechanlage. Die detaillierte Definition mit Ausgangslage, Testschritten und erwartetem Resultat der Testszenarien sind im Anhang C aufgeführt.

Folgendes Protokoll zeigt den Stand der letzten Ausführung der Tests am 21.03.2022:

Szenario	Beschreibung	Resultat
S01	Benachrichtigung versenden - Empfänger konfiguriert	+
S02	Benachrichtigung versenden - Kein Empfänger konfiguriert	+
S03	Benachrichtigung empfangen.	+
S04	Fehler beim Versenden anzeigen.	+
S05	Wiederholen im Fehlerfall bestätigen.	+
S06	Wiederholen im Fehlerfall abbrechen.	+
S07	Audiosignal bei Benachrichtigung.	+
S08	Push Benachrichtigung im Hintergrund.	+
S09	Erinnerungston für nicht quittierte Benachrichtigungen.	+
S10	Start Mobile Client - Nicht angemeldet	+
S11	Start Mobile Client - Angemeldet	+
S12	Anmelden mit korrekten Daten.	+
S13	Anmeldung mit ungültigen Daten.	+
S14	Konfiguration Wählen	+
S15	Abmelden	+
S16	Admin UI - Anmeldung mit korrekten Daten	+
S17	Admin UI - Anmeldung mit ungültigen Daten	+
S18	Admin UI - Konfiguration Verwalten	+
S19	Benachrichtigung vorlesen - Sprachsynthese aktiviert und Benachrichtigung relevant	+
S20	Benachrichtigung nicht vorlesen - Sprachsynthese aktiviert und Benachrichtigung nicht relevant	+
S21	Lokale Einstellung - Sprachsynthese deaktiviert und Benachrichtigung relevant	+
S22	Lokale Einstellung - Sprachsynthese deaktiviert und Benachrichtigung nicht relevant	+
S23	Benachrichtigung verwalten - Relevanz Sprachsynthese kann im Admin UI aktiviert / deaktiviert werden	+
S24	Benachrichtigung empfangen - Änderung an Inhalt einer Benachrichtigung in Admin UI wird sofort angewendet	+
S25	Gegensprechanlage Buttons nach Anmeldung anzeigen	+
S26	Verbindungsaufbau - Gegenüber ist Verfügbar	+
S27	Verbindungsaufbau - Gegenüber ist nicht Verfügbar	+
S28	Verbindungsaufbau - Gegenüber hat Gegensprechanlage deaktiviert	+
S29	Verbindungsaufbau - Benachrichtigungston	+

Szenario	Beschreibung	Resultat
S30	Verbindungsaufbau - Automatische Annahme	+
S31	Unterhaltung 1:1 - Unterhaltung in Echtzeit möglich	+
S32	Unterhaltung 1:n - Unterhaltung in Echtzeit möglich	+
S33	Verbindungsaufbau 1:n	+
S34	Inbox - Vergangene Sprachverbindungen	+
S35	Inbox - Verpasste Sprachverbindungen	+
S36	Inbox - Abgelehnte Unterhaltungen	+
S37	Verbindung trennen durch Empfänger	+
S38	Verbindung trennen durch Initiator	+
S39	Austreten aus Gruppenunterhaltung	+
S40	Konfiguration über Admin UI	+

Alle Testszenarien konnten mit einem positiven Resultat abgeschlossen werden. Das System erfüllt damit die Anforderungen, die im Rahmen dieses Projektes an ein Praxisrufsystem gestellt werden. Das umgesetzte System weist aber durchaus noch Potential zur Weiterentwicklung auf.

Im Bereich Gegensprechanlage, besteht Optimierungspotential für das Entgegennehmen von Anrufen. Das Empfangen von Anrufen ist aktuell nur möglich, wenn die Praxisruf-App aktiv ist. Ist die App nicht aktiv, werden Empfänger über Benachrichtigungen auf verpasste Anrufe hingewiesen. Das Antippen dieser Benachrichtigungen öffnet die Praxisruf-App. Es wird dabei aber kein Anruf entgegengenommen. Für die Weiterentwicklung des Systems wäre es ideal, wenn das Öffnen einer Benachrichtigung für verpasste Anrufe einen direkten Rückruf ermöglicht.

Die Benutzeroberflächen des Mobile Clients erlaubt eine effiziente Bedienung des Systems. Design und User Experience der Applikationen könnten aber weiter verbessert werden. Für die Weiterentwicklung der App sollte die Applikation zusammen mit Praxismitarbeitenden getestet werden. So können diese Rückmeldung zur Bedienung der App geben. Basierend darauf kann die App für Bedürfnisse der Anwenden optimiert werden.

Weiter hat das System noch Lücken die auf das Vorgängerprojekt zurückzuführen sind. Diese Verbesserungen sind ausserhalb des Umfangs dieses Projektes gefallen, sollten für die Weiterentwicklung von Praxisruf aber berücksichtigt werden. Insbesondere das Quittieren von Benachrichtigungen und die Konfigurationsverwaltung könnten verbessert werden.

Das Quittieren von Benachrichtigungen ist mit dem aktuellen System nur lokal auf dem Empfängergerät möglich. Für die Weiterentwicklung des Systems wäre es wünschenswert, dass ein Sender über die Quittierung von Benachrichtigungen informiert wird. So kann er sicher sein, dass die Benachrichtigung empfangen und gelesen wurde.

Das Admin UI ermöglicht heute die Konfiguration des Systems. Die Konfiguration des Systems könnte aber weiter optimiert werden. Heute ist es nicht möglich, die Konfiguration für ein Zimmer mehrfach zu verwenden. Dies bedeutet, dass Praxisadministrierende manche Konfigurationen mehrfach erfassen müssen. Weiter verwendet die Benutzeroberfläche des Admin UI ausschliesslich technische Begriffe für die Konfigurationseinheiten. Für die Weiterentwicklung des Systems sollten sprechende Begriffe in der Benutzeroberfläche verwendet werden. Die Konfiguration des Systems sollte vereinfacht werden, so dass keine doppelten Konfigurationen vorgenommen werden müssen.

Das System bringt noch diverse Lücken aus dem Vorgängerprojekt mit sich und hat Potential sich in den Bereichen Benutzerfreundlichkeit und Stabilität zu verbessern. Trotz dieser Lücken werden die funktionalen Anforderungen, die im Rahmen dieses Projektes an Praxisruf gestellt wurden, erfüllt. Die Funktionstests sind dadurch mit einem positiven Resultat abgeschlossen. Das System kann in der aktuellen Form eingesetzt werden. Für den langfristigen Erfolg des Systems, sollten die bekannten Lücken allerdings bei der Weiterentwicklung adressiert werden.

### 8.2.3 Performancetests

Neben Szenarien für Funktionstests, wurden einfache Tests zur Messung der Performance des Systems definiert. Dazu wurden die Performancekriterien P01 bis P08 definiert. Folgendes Protokoll zeigt den Stand der letzten Ausführung der Tests am 21.03.2022.

Kriterium	Beschreibung	Resultat
P01	Zeit bis Benachrichtigung ankommt im Schnitt < 5s	+
P02	Vorlesen von Benachrichtigung < 5s nach Benachrichtigungston (ohne Cache)	+
P03	Vorlesen von Benachrichtigung < 5s nach Benachrichtigungston (mit Cache)	+
P04	Verbindungsaufbau Sprachverbindung < 5s	+
P05	Verzögerung bei Sprachverbindung klein um kurze Gespräche zu führen	+
P06	Ressourcenverbrauch der Applikation bleibt über Zeit konstant	+
P07	Übertragungsqualität von Sprachverbindungen ist ausreichend für normale Unterhaltungen	+ \ -
P08	Verbindung zur Signaling Instanz bleibt geöffnet und wird bei Fehlern automatisch wiederhergestellt.	+ \ -

Die Performancekriterien P01 bis P06 sind vollständig erfüllt. Die Kriterien P07 und P08 sind mit Einschränkungen erfüllt und zeigen, dass das die Performance des Systems weiter verbessert werden könnte.

Das Kriterium P07 wird mit Einschränkungen erfüllt. Die Qualität der Sprachverbindungen ist ausreichend für eine Gegensprechsanlage. Es ist problemlos möglich Unterhaltungen über die aufgebaute Sprachverbindung zu führen. Probleme können entstehen, wenn eine Verbindung zwischen Endgeräten aufgebaut wird, welche nahe beieinander installiert sind. In diesem Fall können Rückkopplungen entstehen, welche zu einem Echo und schrillen Pfeifton führen kann. Dieses Problem ist durch eine entsprechende Ausrichtung der Geräte vermeidbar. Es schränkt aber ein, wie die Endgeräte des Systems in einer Praxis installiert werden können.

Das Kriterium P08 ist grundsätzlich erfüllt. Wenn die Verbindung zur Signaling Instanz verloren geht, wird eine Fehlermeldung angezeigt. Die Verbindung wird danach automatisch wiederhergestellt. Dies wurde mehrmals über einen Zeitraum von 12h getestet und hat zuverlässig funktioniert. Dabei ist es in Einzelfällen dazu gekommen, dass die Verbindung ohne ersichtlichen Grund verloren ging. In diesen Situation wurde wie erwartet der Fehlerdialog angezeigt. Nach Bestätigung des Dialogs, konnte die Verbindung wiederhergestellt werden. Versuche dieses Verhalten zu reproduzieren waren nicht erfolgreich. Da die Verbindung im Fehlerfall wiederhergestellt werden konnte, wird das Kriterium P08 als erfüllt betrachtet. Für die Weiterentwicklung des Systems, sollte die Verbindungsverwaltung in einem produktionsnahen Umfeld getestet werden. Dabei kann die Performance des Systems beobachtet werden und wenn nötig Korrekturen an der Verwaltung vorgenommen werden.

Letztlich haben Installationstests des Releases für die Praxisruf-App gezeigt, dass die initiale Installation des Systems aufwändig sein kann. In einzelnen Testdurchläufen konnten direkt nach der Installation keine Sprachverbindungen aufgebaut werden. Dieses Problem konnte darauf zurückgeführt werden, dass die Berechtigungen, welche für Netzwerkverbindungen benötigt werden, nicht korrekt angefordert wurden. Im Benutzerhandbuch (Anhang E) wird beschrieben, wie dieses Problem bei der Installation gelöst werden kann. Für die Weiterentwicklung des Systems wäre es wünschenswert, dass alle Berechtigungen automatisch und zwingend beim Start der Applikation angefordert werden.

Die Performancetests konnten mit einem positiven Resultat abgeschlossen werden. Das System kann trotz der bekannten Lücken effizient bedient und als Praxisrufsystem verwendet werden. Die bekannten Lücken sollten aber mit Weiterentwicklung von Praxisruf geschlossen werden.

### 8.3 Fazit

In diesem Kapitel werden die zentralen Herausforderungen während dieser Projektarbeit und die Schlussfolgerungen, die daraus gezogen werden, beschrieben.

Eine zentrale Herausforderung in diesem Projekt waren Konzept und Umsetzung der nativen iOS Applikation. Insbesondere die Integration von Sprachverbindungen und die effiziente Anbindung an Umsysteme waren herausfordernd. Die iOS Standardbibliothek sowie die Bibliotheken der verwendeten Anbieter bieten alle benötigten Komponenten. Die Herausforderung bestand darin, diese Komponenten anzuwenden und Strukturen aufzubauen, welche die effiziente Integration in eine SwiftUI Applikation ermöglichen. Das Erarbeiten, Umsetzen und Dokumentieren dieser Konzepte hat mehr Zeit in Anspruch genommen als erwartet. Schlussendlich hat dieser Aufwand aber einen grossen Mehrwert gebracht. Die Anbindungen an Umsysteme und Peer-To-Peer Sprachverbindungen wurden eigenen Komponenten gekapselt, welche effizient in SwiftUI eingebunden werden können. Im Unterschied zur Entwicklung des Shared Platform Mobile Clients im Vorgängerprojekt, konnte der Aufwand hier mehrheitlich auf konzeptioneller, fachlicher Ebene gehalten werden. Die Anbindung der Schnittstellen und insbesondere die Verwendung von Gerätehardware und Betriebssystemfunktionen wie Push-Benachrichtigungen konnten deutlich einfacher umgesetzt werden. Es sind keine Probleme bezüglich Kompatibilität oder nicht unterstützten Funktionen aufgetreten.

Dieses Projekt hat gezeigt, dass die native Mobile Entwicklung mit SwiftUI die richtige Wahl für die Umsetzung einer App in einem Praxisrufsystem ist. Die native Entwicklung vereinfacht die Integration von Hardware- und Betriebssystemfunktionen massgeblich. Dabei ist es unerlässlich, dass die Konzepte zur Kommunikation mit Umsystemen klar definiert werden. Kommunikationsabläufe und Verantwortlichkeit der internen Komponenten müssen klar definiert werden. Sind diese Voraussetzungen erfüllt, kann von den Vorteilen der nativen Entwicklung profitiert werden.

Die grösste Herausforderung in diesem Projekt war die Umsetzung von Peer-To-Peer Sprachverbindungen mit WebRTC. WebRTC bietet eine Bibliothek für native iOS Entwicklung. Diese implementiert den WebRTC-Standard und bietet alle Komponenten, die zum Aufbau von Sprachverbindungen notwendig sind. Die Bibliothek bietet allerdings keine Entwicklerdokumentation, welche die Verwendung der zur Verfügung gestellten Komponenten dokumentiert. Dies hat die Umsetzung der Lösung deutlich erschwert. Es existieren diverse inoffizielle Beispieldokumentationen und einfache Anleitungen zur Integration von WebRTC in iOS Applikationen. Diese sind allerdings auf ein Minimum reduziert. Sie beinhalten keine Mechanismen zum Verbindungsmanagement und keine Integration in eine grössere Applikation. Die notwendigen Abläufe für Verbindungen konnten basierend auf dem WebRTC Standard erarbeitet werden. Das Konzept zur effizienten Integration einer SwiftUI Applikation wurde mit diesem Projekt erarbeitet. Schlussendlich konnte so eine Lösung umgesetzt werden, die Anforderungen einer Gegensprechanlage im Praxisrufsystem erfüllt. Durch die Verwendung von WebRTC ist die Gegensprechanlage unabhängig von spezifischen Anbietern und bietet maximale Flexibilität für den Betrieb des Systems. Die umgesetzte Signaling Instanz hat keine Abhängigkeiten auf Dienste ausserhalb von Praxisruf und kann bei einem beliebigen Provider betrieben werden. Sprachverbindungen im Mobile Client werden Peer-To-Peer zwischen beteiligten Endgeräten aufgebaut.

Die Erfahrungen in diesem Projekt haben gezeigt, dass WebRTC geeignet ist um Sprachverbindungen in einem Praxisrufsystem umzusetzen. Es ermöglicht eine massgeschneiderte Lösung ohne Bindung an einen externen Dienstleister. Gleichzeitig erschwert es die Umsetzung aufgrund mangelnder Dokumentation. Dieses Problem wird dadurch relativiert, dass WebRTC auf offenen Standards beruht, welche als Grundlage für die Entwicklung dienen können. Weiter bedeutet die Unabhängigkeit von externen Dienstleistern, dass besonders viel Wert auf Performance und Verfügbarkeit gelegt werden muss. Diese können teilweise über den Betrieb der eigenen Serverdienste sichergestellt werden. Sie müssen aber auch auf Applikationsebene behandelt und vor der Produktivsetzung des Systems eingehend getestet werden.

Die letzte Herausforderung, die hier erwähnt wird, betrifft die Planung des Projektes. Das Projekt konnte im Wesentlichen nach dem zu Beginn definierten Projektplan durchgeführt werden. Verzögerungen während der Umsetzung haben aber dazu geführt, dass weniger Zeit für Testing, Fehlerbehebung und Erweiterungen als geplant aufgewendet werden konnte. Regelmässige Absprache mit dem Auftraggeber hat es ermöglicht, Rückmeldungen frühzeitig einzuarbeiten. Der Fokus konnte dabei nur auf die Mindestanforderungen an das System und nicht wie geplant auf Erweiterungen und Optimierungen gelegt werden. Dementsprechend konnten aus dem Vorgängerprojekt bekannte Lücken nicht behoben werden und nicht alle Einschränkungen, die in den Funktions- und Performancetests gefunden wurden, adressiert werden.

Für zukünftige Projekte wird empfohlen, Umsetzungsaufwände detaillierter zu planen und mehr Zeit für das Testen der Anforderungen einzuplanen. Weiter wird empfohlen, das umgesetzte System eingehend in einem produktionsnahen Umfeld zu testen. Dabei sollten Rückmeldungen von den Anwendern eingeholt werden, um weitere Lücken und Verbesserungspotential zu finden. Diese Rückmeldungen können bei der Weiterentwicklung des Systems berücksichtigt werden und erlauben es, das System optimal auf seine Anwendungsgruppen zuzuschneiden.

## 9 Schluss

Im Rahmen dieser Projektarbeit wurden Peer-To-Peer Sprachverbindungen und Sprachsynthese in ein cloudbasiertes Praxisrufsystem integriert. Die umgesetzte Lösung basiert auf dem Praxisrufsystem das im Rahmen des Projektes "IP5 Cloudbasiertes Praxisrufsystem" umgesetzt wurde. Das Praxisrufsystem kann zum Austausch von Informationen in einem Praxismfeld verwendet werden. Kommunikation ist dabei über Benachrichtigungen und Sprachverbindungen zwischen mehreren Teilnehmern möglich.

Das erweiterte System besteht aus einer mobilen Applikation, einer Serverkomponente und einer Webapplikation. Die mobile Applikation wurde neu als native Applikation für iOS entwickelt. Sie ersetzt die Shared Platform Applikation, welche im Rahmen des Vorgängerprojektes entwickelt wurde. Dabei wurden sämtliche bestehende Funktionen und Anbindungen an Umsysteme in die neue Applikation migriert. Mit diesem Projekt neu konzipiert und umgesetzt wurden die Sprachsynthese für empfangene Benachrichtigung sowie die Integration einer Gegensprechanlage über Peer-To-Peer Sprachverbindungen. Die Serverkomponente und Webapplikation wurden erweitert, um diese Funktionen im System verwenden und konfigurieren zu können.

Die mobile Applikation dient als Benutzeroberfläche für das Rufsystem. In der Applikation können über vorkonfigurierte Buttons Sprachverbindungen aufgebaut und Benachrichtigungen versendet werden. Empfangene Benachrichtigungen werden als Push-Benachrichtigungen angezeigt und können auch empfangen werden, wenn die Applikation nicht aktiv ist. Durch die Anbindung von der Dienstleistung Amazon Polly, kann der Inhalt von empfangenen Benachrichtigungen automatisch vorgelesen werden. Die Integration von Sprachverbindungen wurde mit der Technologie WebRTC umgesetzt. Die umgesetzte Lösung erlaubt es, Sprachverbindungen zwischen einem Sender und mehreren Empfängern aufzubauen. Sender und Empfänger können über die geöffnete Verbindung miteinander sprechen. Dadurch kann das Rufsystem als konfigurierbare Gegensprechanlage verwendet werden. Kann ein Empfänger für Sprachverbindungen nicht erreicht werden, wird er mit einer Benachrichtigung über die verpasste Unterhaltung informiert. Sowohl empfangene Benachrichtigungen als auch verpasste und vergangene Anrufe werden gesammelt und in einer Inbox angezeigt. Benachrichtigungen und verpasste Anrufe müssen von Praxismitarbeitenden quittiert werden. Sind unquittierte Elemente in der Inbox, ertönt in regelmässigen Abständen ein Erinnerungston.

Das umgesetzte System deckt die wesentlichen Anforderungen eines cloudbasierten Praxisrufsystems ab. Für eine kommerzielle Nutzung des Systems sind aber zusätzliche Erweiterungen notwendig. Einseitig müssen bekannte Lücken aus dem Vorgängerprojekt geschlossen werden und die Stabilität des Systems in einem produktionsnahem Umfeld verifiziert werden. Weiter muss Praxisruf mandantenfähig werden. Mit dem umgesetzten System ist keine Trennung zwischen verschiedenen Kunden im System möglich. Praxisadministratoren können uneingeschränkt alle bekannten Konfigurationen verwalten. Bei der Auswertung der Konfiguration für das Zustellen von Benachrichtigungen und bei der Signalvermittlung für den Aufbau von Sprachverbindungen wird keine Zuordnung an Benutzer oder Mandat geprüft. Um Praxisruf produktiv bei mehreren Kunden einzusetzen, muss eine Trennung von Mandanten ermöglicht werden. Es wird empfohlen vor der produktiven Nutzung des Systems die Berechtigungsprüfung des Systems zu überarbeiten. Es sollte ein externer Identity Provider angebunden werden und ein Rollensystem entwickelt werden, welches die Verwaltung einzelner Mandate in Praxisruf erlaubt.

Eine weitere Gefahr für die produktive Nutzung des Systems ist, dass es noch nicht über einen längeren Zeitraum in einem produktionsnahen Umfeld getestet wurde. Es wird empfohlen, das System als Prototypen in einem Pilotbetrieb zu testen. So können Rückmeldungen von Praxismitarbeitenden eingeholt werden und die Performance des Systems beobachtet werden. Anhand der Resultate dieser Tests können weitere nötige Optimierungen am System identifiziert werden.

Das Projekt "Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem" konnte erfolgreich abgeschlossen werden. Die erarbeiteten Konzepte bilden eine solide Grundlage für die Archi-

tekur eines cloudbasierten Praxisrufsystems. Das umgesetzte Rufsystem erfüllt die Anforderungen eines cloudbasiertes Praxisrufsystems. Es bildet einen Prototyp, welcher für die Entwicklung eines kommerziell erfolgreichen Praxisrufsystems erweitert werden kann.

## Literaturverzeichnis

- [1] D. Jossen, *21HS-IMVS38: Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem*, 2021.
- [2] J. Villing, K. Zellweger, “Cloudbasiertes Praxisrufsystem,” FHNW - Hochschule für Technik, Techn. Ber., 2021.
- [3] Amazon Webservices, Inc. (31. Okt. 2021). Amazon Polly, Adresse: <https://aws.amazon.com/polly/>.
- [4] Google Developers. (31. Okt. 2021). WebRTC - Echtzeitkommunikation für das Web, Adresse: <https://webrtc.org/>.
- [5] OpenJS Foundation. (4. Jan. 2022). How NativeScript Works, Adresse: <https://v7.docs.nativescript.org/core-concepts/technical-overview>.
- [6] Apple Inc. (9. Nov. 2021). Swift, Adresse: <https://developer.apple.com/swift/>.
- [7] —, (9. Nov. 2021). UIKit, Adresse: <https://developer.apple.com/documentation/uikit/>.
- [8] —, (9. Nov. 2021). SwiftUI, Adresse: <https://developer.apple.com/xcode/swiftui/>.
- [9] —, (9. Nov. 2021). Timer, Adresse: <https://developer.apple.com/tutorials/swiftui/interfacing-with-uikit>.
- [10] Firebase. (9. Nov. 2021). Firebase iOS SDK, Adresse: <https://github.com/firebase/firebase-ios-sdk>.
- [11] Google Developers. (4. Jan. 2022). Set up a Firebase Cloud Messaging client app on Apple platforms, Adresse: <https://firebase.google.com/docs/cloud-messaging/ios/client>.
- [12] Apple Inc. (9. Nov. 2021). AppDelegate, Adresse: <https://developer.apple.com/documentation/uikit/uiapplicationdelegate>.
- [13] —, (8. Jan. 2022). URLRequest, Adresse: <https://developer.apple.com/documentation/foundation/urlrequest>.
- [14] —, (9. Nov. 2021). Timer, Adresse: <https://developer.apple.com/documentation/foundation/timer>.
- [15] —, (9. Nov. 2021). BGTaskScheduler, Adresse: <https://developer.apple.com/documentation/backgroundtasks/bgtaskscheduler>.
- [16] —, (31. Okt. 2021). Speech Synthesis, Adresse: [https://developer.apple.com/documentation/avfoundation/speech\\_synthesis](https://developer.apple.com/documentation/avfoundation/speech_synthesis).
- [17] Amazon Webservices, Inc. (4. Jan. 2022). Amazon Polly iOS Example, Adresse: <https://docs.aws.amazon.com/polly/latest/dg/examples-ios.html>.
- [18] —, (2. März 2021). Amazon Polly SDKs, Adresse: <https://aws.amazon.com/polly/developers/>.
- [19] —, (4. Jan. 2022). Amazon Polly Java Example, Adresse: <https://docs.aws.amazon.com/polly/latest/dg/examples-java.html>.
- [20] —, (). AWS Lambda, Adresse: <https://github.com/aws/amazon-chime-sdk-ios>.
- [21] —, (4. Jan. 2022). Amazon Chime, Adresse: <https://aws.amazon.com/chime/?chime-blog-posts.sort-by=item.additionalFields.createdDate&chime-blog-posts.sort-order=desc>.
- [22] —, (31. Okt. 2021). AWS Chime SDK for iOS, Adresse: <https://github.com/aws/amazon-chime-sdk-ios>.
- [23] —, (12. März 2022). Amazon Chime FAQs, Adresse: <https://aws.amazon.com/chime/faq/>.

- [24] BlogGeek. (4. Jan. 2022). WebRTC Mesh, Adresse: <https://webrtcglossary.com/mesh/>.
- [25] ——, (3. März 2022). WebRTC MCU, Adresse: <https://webrtcglossary.com/mcu/>.
- [26] M. Handley, V. Jacobson, C. Perkins. (3. März 2022). SDP: Session Description Protocol, Adresse: <https://datatracker.ietf.org/doc/html/rfc4566>.
- [27] Mozilla. (3. März 2022). Introduction to WebRTC protocols, Adresse: [https://developer.mozilla.org/en-US/docs/Web/API/WebRTC\\_API/Protocols](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols).
- [28] Ian Hickson, Google Inc. (3. März 2022). WebRTC 1.0: Real-Time Communication Between Browsers, Adresse: <https://www.w3.org/TR/webrtc/>.
- [29] A. Keranen, C. Holmberg, Ericsson J. Rosenberg, jdrosen.net. (3. März 2022). Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal, Adresse: <https://datatracker.ietf.org/doc/html/rfc8445>.
- [30] P. Srisuresh, M. Holdrege, Lucent Technologies. (8. März 2022). IP Network Address Translator (NAT) Terminology and Considerations, Adresse: <https://datatracker.ietf.org/doc/html/rfc2663>.
- [31] Google Developers. (8. März 2022). TURN Server, Adresse: <https://webrtc.org/getting-started/turn-server>.
- [32] E. Rescorla, Mozilla. (8. März 2022). WebRTC Security Architecture, Adresse: <https://www.rfc-editor.org/rfc/rfc8827.html>.
- [33] Apple Inc. (8. Jan. 2022). URLSession, Adresse: <https://developer.apple.com/documentation/foundation/urlsession>.
- [34] ——, (4. März 2022). Result, Adresse: <https://developer.apple.com/documentation/swift/result>.
- [35] ——, (4. März 2022). Decodable, Adresse: <https://developer.apple.com/documentation/swift/decodable>.
- [36] Amazon Webservices, Inc. (8. März 2022). Encryption in Transit, Adresse: <https://docs.aws.amazon.com/polly/latest/dg/encryption-in-transit.html>.
- [37] Apple Inc. (12. Feb. 2022). URLSession.downloadTask, Adresse: <https://developer.apple.com/documentation/foundation/urlsession/1411511-downloadtask>.
- [38] Amazon Web Services. (14. Aug. 2021). AWS Elastic Beanstalk - Getting Started, Adresse: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.CreateApp.html>.
- [39] ——, (14. Aug. 2021). AWS Elastic Beanstalk - Environment properties and other software settings, Adresse: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environments-cfg-softwaresettings.html>.
- [40] ——, (14. Aug. 2021). How can I configure HTTPS for my Elastic Beanstalk environment? Adresse: <https://aws.amazon.com/premiumsupport/knowledge-center/elastic-beanstalk-https-configuration/>.
- [41] ——, (14. Aug. 2021). AWS Elastic Beanstalk - Adding a database to your Elastic Beanstalk environment, Adresse: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html>.
- [42] ——, (14. Aug. 2021). Connecting to a DB instance running the PostgreSQL database engine, Adresse: [https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER\\_ConnectToPostgreSQLInstance.html](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToPostgreSQLInstance.html).
- [43] ——, (14. Aug. 2021). AWS Amplify - Getting started with existing code, Adresse: <https://docs.aws.amazon.com/amplify/latest/userguide/getting-started.html>.
- [44] ——, (14. Aug. 2021). AWS Amplify - Set up custom domains, Adresse: <https://docs.aws.amazon.com/amplify/latest/userguide/custom-domains.html>.

- [45] Google Developers. (14. Aug. 2021). Understand Firebase Projects, Adresse: [https://firebase.google.com/docs/projects/learn-more#setting\\_up\\_a\\_firebase\\_project\\_and\\_registering\\_apps](https://firebase.google.com/docs/projects/learn-more#setting_up_a_firebase_project_and_registering_apps).
- [46] ——, (3. März 2022). WebRTC Source Code Repository, Adresse: <https://webrtc.googlesource.com/src/>.
- [47] Stasel. (24. März 2022). WebRTC-iOS, Adresse: <https://github.com/stasel/WebRTC-iOS>.
- [48] ——, (24. März 2022). WebRTC, Adresse: <https://github.com/stasel/WebRTC>.
- [49] jrendel. (24. März 2022). SwiftKeyChainWrapper, Adresse: <https://github.com/jrendel/SwiftKeychainWrapper>.

## Abbildungsverzeichnis

1.1	Hauptansicht . . . . .	1
1.2	Aktiver Anruf . . . . .	1
1.3	Praxisruf Startseite . . . . .	2
1.4	Systemarchitektur Praxisruf . . . . .	3
2.1	Projektplan . . . . .	4
7.1	Systemkomponenten . . . . .	19
7.2	Entity Relationship Diagramm - Cloudservice . . . . .	22
7.3	Mockup Login . . . . .	23
7.4	Mockup Zimmerwahl . . . . .	23
7.5	Mockup Home . . . . .	24
7.6	Mockup Aktiver Anruf . . . . .	24
7.7	Mockup Inbox . . . . .	25
7.8	Mockup Einstellungen . . . . .	25
7.9	Klassendiagramm PraxisrufApi . . . . .	26
7.10	Klassendiagramm - Mobile Client Services . . . . .	29
7.11	ERD Ausschnitt - Konfiguration Sprachsynthese . . . . .	30
7.12	Klassendiagramm - Modul SpeechSynthesis . . . . .	31
7.13	Sequenzdiagramm - Sprachsynthese auf Systemebene . . . . .	34
7.14	ERD Ausschnitt - Konfiguration Gegensprechanlage . . . . .	36
7.15	Klassendiagramm - Modul Signaling . . . . .	37
7.16	Sequenzdiagramm - Anmeldung und Registrierung . . . . .	39
7.17	Klassendiagramm - Signal . . . . .	40
7.18	Sequenzdiagramm - Verbindungsaufbau mit Offer und Answer Signalen . . . . .	41
7.19	Klassendiagramm - Signaling Schnittstelle in Mobile Client . . . . .	42
7.20	Flowchart - Verbindungsverwaltung . . . . .	43
7.21	Klassendiagramm - CallClient und CallClientDelegate . . . . .	45
7.22	Sequenzdiagramm - Initialisierung Sprachverbindung auf Senderseite . . . . .	46
7.23	Sequenzdiagramm - Initialisierung Sprachverbindung auf Empfängerseite . . . . .	47
8.1	Ansicht Login . . . . .	49
8.2	Ansicht Zimmerwahl . . . . .	49
8.3	Ansicht Home . . . . .	50
8.4	Ansicht Inbox . . . . .	50
8.5	Ansicht Einstellungen . . . . .	51

8.6 Ansicht Aktiver Anruf . . . . .	51
8.7 Hintergrund Benachrichtigung . . . . .	52
8.8 Ansicht Zimmerwahl - Fehlerdialog . . . . .	52
A.1 Aufgabenstellung . . . . .	66

## A Aufgabenstellung



App  
Web

### 21HS\_IMVS38: Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem

Betreuer: [Daniel Jossen](#)

Priorität 1  
Arbeitsumfang: P6 (360h pro Student)

Priorität 2  
---

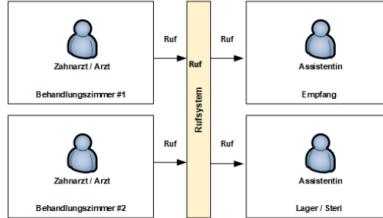
Sprachen: Deutsch

Teamgrösse: Einzelarbeit

---

#### Ausgangslage

Ärzte und Zahnärzte haben den Anspruch in ihren Praxen ein Rufsystem einzusetzen. Dieses Rufsystem ermöglicht, dass der behandelnde Arzt über einen Knopfdruck Hilfe anfordern oder Behandlungsmaterial bestellen kann. Zusätzlich bieten die meisten Rufsysteme die Möglichkeit eine Gegensprechfunktion zu integrieren. Ein durchgeführte Marktanalyse hat gezeigt, dass die meisten auf dem Markt kommerziell erhältlichen Rufsysteme auf proprietären Standards beruhen und ein veraltetes Bussystem oder analoge Funktechnologie zur Signalaübermittlung einsetzen. Weiter können diese Systeme nicht in ein TCP/IP-Netzwerk integriert werden und über eine API extern angesteuert werden.



#### Ziel der Arbeit

Im Rahmen dieser Arbeit soll ein Cloudbasiertes Praxisrufsystem entwickelt werden. Pro Behandlungszimmer wird ein Android oder IOS basiertes Tablet installiert. Auf diese Tablet kann die zu entwickelnde App installiert und betrieben werden. Die App deckt dabei die folgenden Ziele ab:

- Evaluation Frameworks für die Übertragung von Sprachinformationen (1:1 und 1:m)
- Erweiterung SW-Architektur für die Übertragung von Sprachdaten
- Definitoin und Implementierung Text-to-Speech Funktion
- Implementierung Sprachübertragung inklusive Gegensprechfunktion
- Durchführung von Funktions- und Performancetests

#### Problemstellung

Die Hauptproblemstellung dieser Arbeit ist die sichere und effiziente Übertragung von Sprach- und Textmeldungen zwischen den einzelnen Tablets. Dabei soll es möglich sein, dass die App einen Unicast, Broadcast und Multicast Übertragung der Daten ermöglicht. Über eine offene Systemarchitektur müssen die Kommunikationsbuttons in der App frei konfiguriert und parametrisiert werden können.

#### Technologien/Fachliche Schwerpunkte/Referenzen

- Cloud Services (AWS)
- IOS App-Entwicklung (SWIFT)
- Sichere Übertragung von Sprach- und Textmeldungen

#### Bemerkung

Dieses Projekt ist für Joshua Villing reserviert.

Studiengang Informatik/IMVS/Studierendenprojekte 21HS

**Abbildung A.1:** Aufgabenstellung

## B Quellcode

Sämtlicher Quellcode der im Rahmen des Projektes entsteht, wurde mit Git verwaltet. Die Projektbeteiligten haben mit Abschluss des Projektes Zugriff auf Quellcode und erstellte Artefakte erhalten. Weitere Interessenten können Zugriff auf den Quellcode bei Joshua Villing oder Daniel Jossen anfordern.

## C Features und Testszenarien

### F01 - Migration des Mobile Clients

Die Szenarien S01 bis S18 Anhang E des Berichts des Vorgängerprojekts dokumentiert.[2] Diese Szenarien werden für dieses Projekt übernommen und dienen als Testszenarien für F01 Migration des Mobile Clients.

### F02 - Sprachsynthese

Im folgenden sind die Scenarien S19 bis S24 definiert. Diese dienen zur Verifizierung der Umsetzung der Sprachsynthese für Benachrichtigungen.

Scenario S19: Benachrichtigung vorlesen - Sprachsynthese aktiviert und Benachrichtigung relevant

Given: Sprachsynthese für eine Benachrichtigung X ist aktiviert  
And: Sprachsynthese in den App Einstellungen ist aktiviert  
When: Eine Benachrichtigung X wird empfangen  
Then: Der Inhalt der Benachrichtigung wird vorgelesen.

Scenario S20: Benachrichtigung nicht vorlesen - Sprachsynthese aktiviert und Benachrichtigung nicht relevant

Given: Sprachsynthese für eine Benachrichtigung X ist deaktiviert  
And: Sprachsynthese in den App Einstellungen ist aktiviert  
When: Eine Benachrichtigung X wird empfangen  
Then: Der Inhalt der Benachrichtigung wird nicht vorgelesen.

Scenario S21: Lokale Einstellung - Sprachsynthese deaktiviert und Benachrichtigung relevant

Given: Sprachsynthese für eine Benachrichtigung X ist aktiviert  
And: Sprachsynthese in den App Einstellungen ist deaktiviert  
When: Eine Benachrichtigung X wird empfangen  
Then: Der Inhalt der Benachrichtigung wird nicht vorgelesen.

Scenario S22: Lokale Einstellung - Sprachsynthese deaktiviert und Benachrichtigung nicht relevant

Given: Sprachsynthese für eine Benachrichtigung X ist deaktiviert  
And: Sprachsynthese in den App Einstellungen ist deaktiviert  
When: Eine Benachrichtigung X wird empfangen  
Then: Der Inhalt der Benachrichtigung wird vorgelesen.

Scenario S23: Benachrichtigung verwalten - Relevanz Sprachsynthese kann im Admin UI aktiviert / deaktiviert werden

Given: Sprachsynthese für eine Benachrichtigung X ist aktiviert/deaktiviert  
When: Sprachsynthese für die Benachrichtigung X wird über das Admin UI aktiviert/deaktiviert.  
Then: Konfiguration von Benachrichtigung X wird bei Empfang korrekt angewendet.

Scenario S24: Benachrichtigung empfangen - Änderung an Typ in Admin UI wird sofort angewendet

Given: Sprachsynthese für eine Benachrichtigung X ist aktiviert  
 When: Inhalt der Benachrichtigung wird im Admin UI angepasst  
 Then: Bei Empfang der Benachrichtigung, wird der angepasste Inhalt der Benachrichtigung vorgelesen.

### F03 - Gegensprechanlage

Scenario S25: Gegensprechanlage Buttons nach Anmeldung anzeigen

Given: Konfiguration X ist im Clouservice erfasst  
 And: Für Konfiguration X wurden Buttons für Sprachverbindungen definiert.  
 When: Benutzer meldet sich an und wählt Konfiguration X  
 Then: Die konfigurierten Buttons werden geladen und angezeigt

Scenario S26: Verbindungsaufbau - Gegenüber ist Verfügbar

Given: Ziel der Sprachverbindung ist beim Signaling Server registriert.  
 And: Ziel der Sprachverbindung hat das Empfangen von Anrufen in der lokalen Konfiguration aktiviert.  
 When: Button für Sprachverbindung wird aktiviert  
 Then: Die Sprachverbindung wird aufgebaut und die Ansicht für Aktive Anrufe auf beiden Clients angezeigt.

Scenario S27: Verbindungsaufbau - Gegenüber ist nicht Verfügbar

Given: Ziel der Sprachverbindung ist nicht beim Signaling Server registriert  
 When: Button für Sprachverbindung wird aktiviert  
 Then: Ziel der Sprachverbindung erhält eine Benachrichtigung für verpassten Anruf  
 Then: Ziel der Sprachverbindung sieht einen Eintrag für den verpassten Anruf in der Inbox.  
 Then: Dem Initiator der Verbindung wird angezeigt, dass das Ziel nicht erreicht wurde.

Scenario S28: Verbindungsaufbau - Gegenüber hat Gegensprechanlage deaktiviert

Given: Ziel der Sprachverbindung ist beim Signaling Server registriert  
 When: Button für Sprachverbindung wird aktiviert  
 Then: Ziel der Sprachverbindung erhält eine Benachrichtigung für verpassten Anruf  
 And: Ziel der Sprachverbindung sieht einen Eintrag für den verpassten Anruf in der Inbox.  
 And: Dem Initiator der Verbindung wird angezeigt, dass das Ziel nicht erreicht wurde.

Scenario S29: Verbindungsaufbau - Benachrichtigungston

Given: Empfang von Sprachverbindungen ist aktiviert  
 When: Eine Sprachverbindung wird empfangen  
 Then: Ein Benachrichtigungston ertönt, bevor die Verbindung angenommen wird.

- Scenario S30:** Verbindungsauftbau - Automatische Annahme
- Given:** Empfang von Sprachverbindungen ist aktiviert  
**When:** Eine Sprachverbindung wird empfangen.  
**Then:** Die Verbindung wird automatisch und ohne weitere Benutzereingaben geöffnet.
- Scenario S31:** Unterhaltung 1:1 - Unterhaltung in Echtzeit möglich
- Given:** Sprachverbindung zwischen zwei Teilnehmern wurde initialisiert  
**When:** Die Sprachverbindung aufgebaut ist.  
**Then:** Können beide Teilnehmer in Echtzeit kommunizieren.
- Scenario S32:** Unterhaltung 1:n - Unterhaltung in Echtzeit möglich
- Given:** Alle Ziele einer Sprachverbindung sind beim Signaling Service registriert  
**And:** Alle Ziele einer Sprachverbindung haben den Empfang von Anrufen aktiviert.  
**When:** Eine Sprachverbindung zu mehreren Teilnehmern aufgebaut wird  
**Then:** Können der Initiator mit allen Teilnehmern in Echtzeit kommunizieren.  
**Then:** Kann jedes Ziel mit dem Initiator Echtzeit kommunizieren.  
**Then:** Können die Ziele nicht direkt kommunizieren.
- Scenario S33:** Verbindungsauftbau 1:n
- Given:** Alle Ziele einer Sprachverbindung sind beim Signaling Service registriert  
**And:** Alle Ziele einer Sprachverbindung haben den Empfang von Anrufen aktiviert.  
**When:** Eine Sprachverbindung zu mehreren Teilnehmern aufgebaut wird  
**Then:** Der Initiator sieht die Verbindung und den Status aller Ziele  
**Then:** Jedes Ziel verhält sich als ob es Teil einer Einzelunterhaltung ist (S26-S30).
- Scenario S34:** Inbox - Vergangene Sprachverbindungen
- Given:** Eine Sprachverbindung wurde empfangen.  
**When:** Die Sprachverbindung wird beendet.  
**Then:** Es ist ein Eintrag für die Verbindung in der Inbox zu sehen.
- Scenario S35:** Inbox - Verpasste Sprachverbindungen
- Given:** Das Ziel einer Sprachverbindung ist nicht erreichbar.  
**When:** Es wird versucht eine Sprachverbindung zum Ziel aufzubauen.  
**Then:** Es ist ein Eintrag für die verpasste Verbindung in der Inbox zu sehen.
- Scenario S36:** Inbox - Abgelehnte Unterhaltungen
- Given:** Das Ziel einer Sprachverbindung ist nicht erreichbar.  
**And:** Das Ziel einer hat den Empfang von Anrufen lokal deaktiviert.  
**When:** Es wird versucht eine Sprachverbindung zum Ziel aufzubauen.  
**Then:** Es ist ein Eintrag für die verpasste Verbindung in der Inbox zu sehen.

Scenario S37: Verbindung trennen durch Empfänger

Given: Eine Sprachverbindung zwischen beliebig vielen Teilnehmern wurde aufgebaut.  
When: Das Ziel der Verbindung beendet die Verbindung.  
Then: Die Verbindung wird auf beiden Seiten beendet.

Scenario S38: Verbindung durch Initiator

Given: Eine Sprachverbindung zwischen zwei Teilnehmern wurde aufgebaut.  
When: Der Initiator der Verbindung beendet die Verbindung.  
Then: Die Verbindung wird auf beiden Seiten beendet.

Scenario S39: Austreten aus Gruppenunterhaltung

Given: Eine Sprachverbindung zwischen mehr als zwei Teilnehmern wurde aufgebaut.  
When: Eines der Ziele beendet die Verbindung.  
Then: Die Verbindung wird für dieses Ziel beendet.  
And: Die Verbindungen aller weiteren Teilnehmer bleiben offen.

Scenario S40: Konfiguration über Admin UI

Given: Admin ist angemeldet  
When: Admin UI wird aufgerufen  
Then: Alle konfigurierten Buttons für Sprachverbindungen werden angezeigt.  
And: Neue Buttons für Sprachverbindungen können erstellt werden  
And: Bestehende Buttons für Sprachverbindungen können verändert werden  
And: Bestehende Buttons für Sprachverbindungen können gelöscht werden

## D Installationsanleitung

In diesem Kapitel wird beschrieben wie Praxisruf mit Amazon Webservices betrieben werden kann. Die hier aufgeführten Anleitungen sind aus dem Projektbericht "IP5 Cloudbasiertes Praxisrufsystem" [2] übernommen und wo nötig ergänzt. Sämtliche Anleitungen behandeln ausschliesslich eine Neuinstallation des Systems und beinhalten keine Migrationsanleitung auf Basis des Vorgängerprojekts.

### Firebase Messaging

Im Folgenden wird beschrieben wie Firebase Cloud Messaging konfiguriert werden muss, um Benachrichtigungen in Praxisruf zu ermöglichen. Die Konfiguration von Firebase Cloud Messaging muss vor Installation von Cloudservice und Mobile Client vorgenommen werden. Diese Anleitung hat sich im Vergleich zum Vorgängerprojekt nicht verändert und ist direkt aus dem Anhang D des Projektberichts "IP5 Cloudbasiertes Praxisrufsystem" [2] übernommen.

1. Öffnen Sie die *Firebase Console* und erstellen Sie ein neues Projekt.
2. Navigieren Sie in das Menü "Project Settings".
3. Navigieren Sie zum Tab "Service Accounts".
4. Klicken Sie den Button "Generate new private Key" und bestätigen Sie die Eingabe.
5. Speichern Sie die generierte Datei an einem Ort Ihrer Wahl.
6. Kopieren Sie den Dateiinhalt und Benutzen Sie ein Werkzeug Ihrer Wahl<sup>3</sup> um den Inhalt mit Base64 zu kodieren.
7. Speichern Sie den Base64 kodierten Dateiinhalt an einem Ort Ihrer Wahl. Sie werden diesen Wert für die Konfiguration des Cloud Services benötigen.
8. Öffnen Sie erneut die *Firebase Console*
9. Navigieren Sie in das Menü "Project Settings".
10. Erstellen Sie für die gewünschte Plattform (iOS, Android oder Web app) eine Anwendung via "Add App".
11. Verwenden Sie für den Android package name bzw. die iOS bundle ID den package\_name des Mobile Clients. (ch.fhnw.ip5.praxisintercom.client)
12. Vergeben Sie einen beliebigen App-Nickname.
13. Laden Sie nach dem Registrieren die Konfigurationsdatei GoogleService-Info.plist (iOS) oder google-services.json (Android) herunter.
14. Legen Sie die Konfigurationsdatei in einem Verzeichnis Ihrer Wahl ab.

Mehr Informationen zu Firebase und der Integration von Firebase Projekten finden Sie in der offiziellen Dokumentation.[45]

---

<sup>3</sup>z.B. <https://www.base64decode.org/>

## Cloudservice

### *Installation mit Amazon Webservices*

Im Folgenden wird beschrieben wie die Cloud Service Applikation mit AWS betrieben werden kann. Diese Anleitung wurde im Vergleich zum Vorgängerprojekt erweitert um die Konfiguration von Gegen sprechanlage und Sprachsynthese zu erweitert. Dazu wurde die Anleitung aus dem Anhang D des Projektberichts "IP5 Cloudbasiertes Praxisrufsystem" [2] erweitert.

1. Stellen Sie sicher, dass der Quellcode der Cloud Service Applikation in einem Git Repository bei einem der Anbieter Github, Gitlab, BitBucket oder AWS CodeCommit zur Verfügung steht.
2. Erstellen Sie einen mit AWS ein Elastic Beanstalk Environment.
  - (a) Folgen Sie dazu der *offiziellen Anleitung*[38]
  - (b) Wählen Sie unter Plattform "Java" und die dazugehörigen Standardeinstellungen.
  - (c) Wählen Sie unter Application Code "Sample Application".
3. Erstellen Sie mit AWS RDS eine Datenbank für die Cloud Service Applikation
  - (a) Öffnen Sie die Beanstalk Console.
  - (b) Folgen Sie der *offiziellen Anleitung*[41] um eine Relationale Datenbank an das Beanstalk Environment anzubinden.
  - (c) Wählen Sie als Datenbank Engine "postgres" in der Version 13.3.
  - (d) Wenn sie oben genannter Anleitung folgen, ist keine weitere Konfiguration für die Datenbankanbindung im Cloud Service nötig. Sollten Sie wählen, die Datenbank auf eine andere Art zu Betreiben müssen in Schritt 4 die UmgebungsvARIABLEN RDS\_HOSTNAME, RDS\_PORT, RDS\_DB\_NAME, RDS\_USERNAME und RDS\_PASSWORD mit den entsprechenden Werten konfiguriert werden.
4. Um die Anbindung an Amazon Polly nutzen zu können, muss ein Benutzer für den Zugriff definiert und berechtigt werden.
  - (a) Erstellen Sie einen IAM Benutzer für den Zugriff auf Amazon Polly. Folgen Sie dazu der *offiziellen Anleitung*.
  - (b) Stellen Sie sicher, dass der Benutzer die Rolle *AmazonPollyFullAccess* zugewiesen hat.
  - (c) Erstellen Sie einen Access Key für den erstellten Benutzer. Folgen Sie dazu der *offiziellen Anleitung*.
  - (d) Speichern Sie den generierten Access Key ID und Access Secret Key an einem sicheren Ort. Sie werden für die weitere Konfiguration benötigt.
5. Definieren Sie die nötigen Umgebungsvariablen für die Cloud Service Applikation:
  - (a) Folgen Sie der *offiziellen Anleitung*[39] um die nötigen Umgebungsvariablen zu setzen:
  - (b) Name: FCM\_CREDENTIALS, Wert: Firebase Credentials mit Base 64 Encoded<sup>4</sup>
  - (c) Name: SPRING\_PROFILES\_ACTIVE, Wert: aws.
  - (d) Name: ADMIN\_ORIGIN, Wert: Admin UI Domain
  - (e) Name: JWT\_SECRET\_KEY, Wert: Zufälliger 64 Bit String<sup>5</sup>
  - (f) Name: AWS\_ACCESS\_KEY, Wert: Access Key ID, welcher in Schritt 4 erstellt wurde.
  - (g) Name: AWS\_SECRET\_KEY, Wert: Access Secret Key, welcher in Schritt 4 erstellt wurde.
  - (h) Name: NOTIFICATION\_TYPE\_FOR\_UNAVAILABLE, Wert: '63d530ab-48af-4597-a9fd-2fb4c9700c55'. Oder die ID des NotificationTypes, welcher für verpasste Anrufe verwendet werden soll.
6. Konfigurieren Sie AWS CodeBuild um die Cloud Service Applikation mit AWS bauen zu können.
  - (a) Öffnen Sie die AWS Console und wählen Sie unter Services "Code Pipeline" aus.
  - (b) Wählen Sie die Option "Create build project".
  - (c) Geben Sie unter Project Configuration einen passenden Namen ein.

<sup>4</sup>Siehe Installationsanleitung Firebase Messaging

<sup>5</sup><https://www.grc.com/passwords.htm>

- (d) Wählen Sie unter Source den gewünschten Anbieter und geben Sie die Repository URL sowie die gewünschte Version (Branch Name) ein.
  - (e) Wählen Sie unter Buildspec "Use a buildspec file". Die projektspezifische Build Konfiguration ist in der Cloud Service Applikation bereits enthalten.
  - (f) Bestätigen Sie die Eingaben.
7. Konfigurieren Sie AWS CodePipeline um die Cloud Service Applikation zu installieren:
- (a) Öffnen Sie die AWS Console und wählen Sie unter Services "Code Pipeline" aus.
  - (b) Wählen Sie die Option "Create New Pipeline".
  - (c) Geben Sie in Schritt 1 einen Namen für die Pipeline an und wählen Sie "Next".
  - (d) Wählen Sie in Schritt 2 gewünschten Anbieter und folgen sie dem Wizard um das Git Repository des Cloud Services anzubinden.
  - (e) Wählen Sie in Schritt 3 AWS CodeBuild und das CodeBuild Projekt welches Sie in Schritt 3 erstellt haben.
  - (f) Wählen Sie in Schritt 4 AWS Elastic Beanstalk und die in Schritt 2 definierten Instanzen.
  - (g) Bestätigen Sie die Eingaben.
8. Stellen Sie sicher, dass die installierte Cloud Service Applikation über HTTPS erreichbar ist. Dies ist für die Kommunikation mit Mobile Client Instanzen zwingend notwendig.
- (a) Folgen Sie dazu der offiziellen Anleitung *offiziellen Anleitung*[40].
  - (b) Am Cloud Service sind dabei keine Änderungen nötig.
9. Stellen Sie sicher, dass die in Schritt 7 konfigurierte Pipeline erfolgreich ausgeführt wurde. Durch das erste Starten der Applikation wird das Datenbankschema automatisch initialisiert.
10. Als letzter Schritt müssen zwei Scripts manuell auf der Datenbank ausgeführt werden. Gehen Sie dazu wie folgt vor:
- (a) Richten Sie den Datenbankzugriff auf die in Schritt 3 erstellte Datenbank gemäss der *offiziellen Anleitung*[42]. ein.
  - (b) Wählen Sie ein Passwort
  - (c) Erstellen Sie ein Hash des gewählten Passwort mit dem "bcryptAlgorithmus".<sup>6</sup>
  - (d) Passen Sie die Parameter in folgenden Script an und führen es auf der Datenbank aus:

```

1 insert into
2   praxis_intercom_user (id, name, password, role, user_name)
3 values
4   (<uuid>, 'admin', <encoded password>, 'admin', <username>);

```

**Listing 1:** createadmin.sql

- (e) Erstellen Sie die Konfiguration für Benachrichtigung bei verpassten Anrufen in der Gegen sprechanlage, indem Sie folgendes Script ausführen:

```

1 INSERT INTO
2   notification_type (id, body, display_text, text_to_speech, title, type,
3   version)
4 VALUES
5   ('63d530ab-48af-4597-a9fd-2fb4c9700c55', 'Verpasster Anruf',
6   ' ', false, 'Verpasster Anruf', 'Verpasster Anruf', 0);

```

**Listing 2:** missedcallnotification.sql

---

<sup>6</sup><https://bcrypt-generator.com/>

### *Lokale Installation*

Die folgende Anleitung beschreibt, wie der Cloudservice lokal gestartet werden kann. Diese Art den Cloudservice zu installieren sollte nur zu Testzwecken verwendet werden.

1. Stellen Sie sicher, dass auf Ihrem System Java in der Version 13 oder neuer installiert ist.
2. Installieren Sie PostgreSQL auf Ihrem System und erstellen Sie eine Datenbank für den Cloudservice. Verwenden Sie die folgenden Informationen für die Konfiguration:
  - (a) Port: 5433
  - (b) Benutzer: admin
  - (c) Passwort: secret
  - (d) Name der Datenbank: intercom\_configuration\_db\_local
3. Führen Sie die Skripts aus Schritt 10 aus dem vorhergehenden Abschnitt auf der erstellten Datenbank aus.
4. Stellen Sie sicher, dass die Umgebungsvariablen aus Schritt 5 der Anleitung für Installation mit AWS auf Ihrem System gesetzt sind.
5. Kopieren Sie die .jar Datei des Cloudservices auf Ihr System
6. Öffnen Sie ein Terminal im Verzeichnis in welchen die .jar Datei abgelegt ist.
7. Führen Sie den Befehl ”java -jar -Dspring.profiles.active=local ./<filename>.jar” aus.
8. Der Cloudservice wird gestartet und ist danach unter der localhost:5000 erreichbar.
9. Die API Dokumentation des Cloudservice ist unter localhost:5000/swagger-ui.html zu finden.

### *Cloudservice Release bauen*

Mit dem Sourcecode des Cloudservice wurde eine ausführbare .jar-Datei basierend auf dem aktuellen Stand des Cloudservice veröffentlicht. Sie finden dieses im Verzeichnis release im Rootverzeichnis des Cloudservice-Projektes. Dieses Image kann, wie im vorhergehenden Abschnitt beschrieben installiert werden.

Um einen neuen Release zu bauen, gehen Sie wie folgt vor:

1. Öffnen Sie ein Terminal im Rootverzeichnis des Cloudservice-Projektes.
2. Setzen Sie den Befehl ”./gradlew build” ab.
3. Nach Abschluss des Befehls finden Sie die ausführbare .jar Datei im Verzeichnis ./praxisruf-app/build/libs (Relativ zum Rootverzeichnis des Cloudservice-Projektes).

## Admin UI

### *Installation mit Amazon Webservices*

Im Folgenden wird beschrieben wie die Admin UI Applikation mit AWS betrieben werden kann. Diese Anleitung hat sich im Vergleich zum Vorgängerprojekt nicht verändert und ist direkt aus dem Anhang D des Projektberichts "IP5 Cloudbasiertes Praxisrufsystem" [2] übernommen.

1. AWS Amplify Service aufsetzen:
  - (a) Amazon Webservice unterstützt die Anbindung von Github, Gitlab, BitBucket und AWS CodeCommit. Stellen Sie sicher, dass der Quellcode der Admin UI Applikation in einem Git Repository zur Verfügung steht.
  - (b) Folgen Sie den Schritten in der *offiziellen Anleitung*.[43]
2. Verbindung zum Cloud Service konfigurieren:
  - (a) Öffnen sie die AWS Amplify Konsole für die in Schritt 1 erstellte Applikation.
  - (b) Wählen Sie den Menüpunkt "Environment Variables"
  - (c) Erstellen Sie eine neue Variable mit dem Namen REACT\_APP\_BACKEND\_BASE\_URI. Setzen Sie als Wert dafür die Domain, unter welcher der Cloud Service erreichbar ist.
3. Konfigurieren Sie eine Domain für die Admin UI Applikation.
  - (a) Folgen Sie dazu den Schritten in der *offiziellen Anleitung*.[44] folgen.

### *Lokale Installation*

Das Admin UI kann lokal mit einem Webserver betrieben werden. Diese Art das Admin UI zu betreiben sollte nur zu Testzwecken verwendet werden.

1. Stellen Sie sicher, dass npm<sup>7</sup> und Node.js<sup>8</sup> auf Ihrem System installiert sind.
2. Stellen Sie sicher, dass der Webserver "serve"<sup>9</sup> auf Ihrem System installiert ist.
3. Kopieren Sie die Dateien des Releases in ein Verzeichnis auf Ihrem System.
4. Öffnen Sie ein Terminal in diesem Verzeichnis.
5. Führen Sie den Befehl "serve -s ." aus.
6. Das Admin UI ist danach unter localhost:3000 erreichbar. Wenn Sie den Release Build, der mit der Abgabe des Projektes veröffentlicht wurde verwenden, muss der Cloudservice unter localhost:5000 erreichbar sein.

### *Admin UI Release bauen*

Mit dem Sourcecode des Admin UIs wurde ein Release Build basierend auf dem aktuellsten Stand des Projektes geliefert. Dieser kann, wie im vorhergehenden Abschnitt beschrieben installiert werden.

Um einen neuen Release Build auszuführen, gehen Sie wie folgt vor:

1. Stellen Sie sicher, dass npm und Node.js auf Ihrem System installiert sind.
2. Stellen Sie sicher, dass folgende UmgebungsvARIABLE gesetzt ist:
  - (a) Name: REACT\_APP\_BACKEND\_BASE\_URI Wert: Basis URL des Cloudservice.
3. Öffnen Sie ein Terminal im Rootverzeichnis des Admin UI Projektes.
4. Führen Sie den Befehl "npm install" aus.
5. Führen Sie den Befehl "npm run build" aus.
6. Die Releasedateien des Admin UIs finden sich danach im Verzeichnis ./build unter dem Rootverzeichnis des Projektes.

---

<sup>7</sup><https://www.npmjs.com/>

<sup>8</sup><https://nodejs.org/en/>

<sup>9</sup><https://www.npmjs.com/package/serve>

## Mobile Client

Dieses Kapitel beschreibt wie die native iOS Applikation auf Geräten installiert werden kann.

Mit dem Quellquode des Mobile Clients ist ein deploybares Image veröffentlicht. Dieses beinhaltet die Credentials und Konfigurationen für die Infrastruktur die während diese Projektarbeit aufgebaut wurden. Das Image ist im Verzeichnis release unter dem Rootverzeichnis des Mobile Client-Projektes abgelegt. Solange diese Infrastruktur besteht und verwendet wird, kann das veröffentlichte Image verwendet werden, um Praxisruf zu betreiben.

Das veröffentlichte Image und Images die neu gebaut werden, können wie folgt installiert werden:

### *Installation eines bestehenden Image*

1. Öffnen Sie das Projekt *praxisruf-ios-mobile-client* in die XCode Entwicklungsumgebung.
2. Öffnen Sie die Deviceliste innerhalb von Xcode. (cmd + shift + 2)
3. Installieren Sie das .ipa File des Images via Drag and Drop auf dem IPad.

### *Mobile Client Image bauen*

Die Integration von Firebase Cloud Messaging in den Mobile Client muss zur Buildzeit passieren. Dies bedeutet, dass die Applikation neu gebaut werden muss. Dazu gehen Sie wie folgt vor:

1. Öffnen Sie das Projekt *praxisruf-ios-mobile-client* in die XCode Entwicklungsumgebung.
2. Öffnen Sie die Projekteinstellungen (Doppelklick auf das Topverzeichnis *praxisruf-ios-mobile-client* in XCode)
3. Navigieren Sie zu Build Settings > User Defined
4. Setzten Sie den Wert für BASE\_URL\_HTTPS auf die URL unter welcher die HTTP API des Cloudservice erreichbar ist. Dies entspricht standardmäßig <serverUrl>/api
5. Setzten Sie den Wert für BASE\_URL\_WSS unter welcher die WebSocket API des Cloudservice erreichbar ist. Dies entspricht standardmäßig <serverUrl>
6. Wenn Sie die Instanz des Cloudservice unter "www.praxisruf.ch" verwenden, müssen Sie die Werte nicht anpassen.
7. Speichern Sie die Konfigurationsdatei aus Schritt 13 der Firebase Cloud Messaging Installationsanleitung im XCode Projekt unter *praxisruf-ios-mobile-client/Resources/GoogleService-Info.plist* ab.
8. Führen Sie den Build der Applikation aus. (shift + command + R).
9. Wählen Sie in der Top Menüleiste von XCode Product > Archive.
10. Warten Sie bis die Aktion beendet hat und der Dialog *Archives* sich öffnet.
11. Wählen Sie im Dialog *Distribute App*
12. Wählen Sie die Option *Ad Hoc* und fahren mit *next* fort.
13. Aktivieren Sie die Option *Rebuild from Bitcode* deaktivieren Sie alle anderen Optionen und fahren mit *next* fort.
14. Wählen Sie *Automatically manage signing* und fahren mit *next* fort.
15. Bestätigen Sie mit *Export*
16. Geben Sie das Verzeichnis, an in welches die Applikation exportiert werden soll.
17. Die in diesem Verzeichnis abgelegte .ipa Datei kann nun, wie in *Installation eines bestehenden Image* beschrieben, auf Geräten installiert werden.

## E Benutzerhandbuch

### Mobile Client

#### Anmeldung und Konfiguration

1. Mobile Client Applikation öffnen
2. Anmeldedaten eingeben und bestätigen
3. Gewünschte Konfiguration auswählen

#### Benachrichtigungen

1. In Navigationsleiste (unten) "Home" auswählen
2. Im Bereich "Benachrichtigung" Button mit dem gewünschten Titel antippen
3. Die Benachrichtigung wird versendet

#### Sprachverbindungen

1. In Navigationsleiste (unten) "Home" auswählen
2. Im Bereich "Gegensprechanlage" Button mit dem gewünschten Titel antippen
3. Die Sprachverbindung wird aufgebaut.
4. In der geöffneten Ansicht sehen Sie alle Teilnehmer und deren Status.
5. Die grauen Buttons in der Ansicht geben die Möglichkeit, Mikrofon und Lautsprecher stummzuschalten.
6. Mit dem roten Button in der Ansicht kann die Verbindung getrennt werden.

#### Inbox

1. In Navigationsleiste (unten) "Inbox" auswählen
2. In dieser Ansicht sehen Sie alle empfangenen noch nicht quittierten Benachrichtigungen
3. Zum Quittieren eines Eintrages den Eintrag nach Links wischen.

#### Lokale Einstellungen ändern

1. In Navigationsleiste (unten) "Einstellungen" auswählen
2. Über die Schaltfläche "Benachrichtigungen vorlesen" kann das Vorlesen für alle Benachrichtigungen deaktiviert werden.
3. Über die Schaltfläche "Anrufe empfangen" können eingehende Anrufe automatisch abgelehnt werden.
4. Über den Button logout können Sie sich aus der Applikation abmelden.

#### Abmeldung

1. In Navigationsleiste (unten) "Einstellungen" auswählen
2. Button "Abmelden" antippen.

#### Berechtigungen nach Installation

Die Praxisruf-App benötigt mehrere lokale Berechtigungen. Um sicherzustellen, dass alle Funktionen zur Verfügung stehen, müssen diese vom Benutzer bestätigt werden. Zu diesem Zweck müssen alle Funktionen direkt nach Installation einmal verwendet werden. Nachdem diese Berechtigungen einmal erteilt sind, müssen Sie nicht erneut bestätigt werden. Gehen Sie dazu wie folgt vor:

1. Öffnen Sie die Applikation und melden Sie sich an.
2. Bestätigen Sie, dass Praxisruf Benachrichtigungen anzeigen darf.
3. Starten Sie anschliessend einen Anruf zu einem anderen Gerät.

4. Bestätigen Sie ebenfalls, dass Praxisruf Geräte im Lokalen Netzwerk entdecken darf. Dies ist notwendig, um die Verbindung zu anderen Geräten herzustellen.
5. Bestätigen Sie, dass Praxisruf auf Mikrofon und Lautsprecher zugreifen darf. Dies ist notwendig um Gespräche über die aufgebauten Verbindungen zu führen.
6. Es ist möglich, dass nicht alle Dialoge für die benötigten Berechtigungen angezeigt werden. Sollten Verbindungen nicht hergestellt werden können, gehen Sie wie folgt vor:
  - (a) Führen Sie mehrere Verbindungsversuche aus. Wenn die Dialoge nach mehreren Versuchen nicht angezeigt werden, fahren Sie mit b) fort.
  - (b) Starten Sie die Applikation neu und führen anschliessend weitere Versuche aus. Wenn die Dialoge weiterhin nicht angezeigt werden, fahren Sie mit c) fort.
  - (c) Beenden Sie die Applikation und trennen alle Netzwerkverbindungen des Gerätes. Verbinden Sie das Gerät erneut mit dem Netzwerk. Starten Sie die Applikation anschliessend neu und führen Sie weitere Verbindungsversuche aus.

## **Netzwerk**

Um sicherzustellen, dass Daten ausgetauscht werden können, müssen Sie auf folgendes achten:

1. Benachrichtigungen können nur ausgetauscht werden, wenn die Geräte eine Verbindung ins Internet haben.
2. Anrufe können nur zwischen Geräten, die im selben lokalen Netzwerk sind geführt werden.

## **Admin UI**

### **Anmeldung**

1. Nach Installation des Systems Anmeldedaten und URL des Admin UI beim Betreiber einholen
2. Admin UI im Browser öffnen
3. Anmeldedaten eingeben und bestätigen

### **Konfiguration verwalten**

1. Melden Sie sich im Admin UI an.
2. Wählen Sie auf der Linken Seite die Kategorie, die Sie verwalten möchten.
3. Auf dieser Seite können Sie nun Einträge zur gewählten Kategorie verwalten:
  - Klicken Sie oben rechts auf die Schaltfläche “Create” um einen neuen Eintrag zu erstellen.
  - Klicken Sie auf einen Eintrag in der Liste, um ihn zu bearbeiten.
  - Klicken Sie die Checkbox auf der Linken Seite eines Eintrages und dann auf die Schaltfläche “löschen”, um einen Eintrag zu löschen.

### **Praxisruf konfigurieren**

1. Melden Sie sich im Admin UI an.
2. Erfassen Sie in der Kategorie “User” mindestens einen Benutzer.
3. Erfassen Sie in der Kategorie “Client” pro Endgerät in Ihrem System einen Eintrag und weisen Sie einem Benutzer zu.
4. Erfassen Sie in der Kategorie “Notification Types” alle Benachrichtigungen die Sie zur Verfügung stellen möchten.
5. Erfassen Sie in der Kategorie “CallTypes” alle Sprachverbindungen die Sie zur Verfügung stellen möchten.
6. Erfassen Sie in der Kategorie “Configurations” pro Gerät einen Eintrag und weisen Sie ihn dem entsprechenden Client zu.
7. Unter Notification Types können die Benachrichtigungen auswählen, die auf diesem Gerät zum Versenden verfügbar sind.

8. Unter Rule Parameters können Sie Regeln definieren, welche Benachrichtigungen diesem Client zugestellt werden.
  - Mit dem Rule Type “Sender”, werden alle Benachrichtigungen vom angegebenen Sender Empfangen.
  - Mit dem Rule Type “NotificationType”, werden alle Benachrichtigungen mit dem angegebenen Typ empfangen.

### **Benachrichtigung für verpasste Anrufe**

Die Benachrichtigung für verpasste Anrufe kann im Admin UI erfasst und verändert werden. Eine Benachrichtigung mit Standardwerten wurde bei der Installation erfasst (vgl. Installationsanleitung). Diese kann wie folgt angepasst werden:

1. Melden Sie sich im Admin UI an.
2. Finden Sie unter ”NotificationTypes” die Benachrichtigung mit dem Title ”Verpasster Anruf”.
3. Sie können diese Benachrichtigung frei bearbeiten. Die entsprechenden Änderungen werden für alle folgenden Benachrichtigungen für verpasste Anrufe angewendet.

Sie können weiter eine beliebige andere Benachrichtigung für verpasste Anrufe verwenden. Dazu gehen Sie wie folgt vor:

1. Melden Sie sich im Admin UI an.
2. Erstellen Sie einen neuen NotificationType und geben Sie die gewünschten Werte ein.
3. Speichern Sie den NotificationType.
4. Öffnen Sie die Detailansicht für den NotificationType.
5. Kopieren Sie die technische Id des NotificationType aus der URL im Browserfenster. Die id ist der letzte Teil der URL.
6. Aktualisieren Sie die Konfiguration des Cloudservice und setzen den Wert der Umgebungsvariable NOTIFICATION\_TYPE.FOR\_UNAVAILABLE auf die kopierte Id.
7. Starten Sie den Cloudservice neu. Anschliessend wird die konfigurierte Benachrichtigung für alle folgenden verpassten Anrufe verwendet.

## F Ehrlichkeitserklärung

«Hiermit erkläre ich, die vorliegende Projektarbeit Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem selbständig und nur unter Benutzung der angegebenen Quellen verfasst zu haben. Die wörtlich oder inhaltlich aus den aufgeführten Quellen entnommenen Stellen sind in der Arbeit als Zitat bzw. Paraphrase kenntlich gemacht. Diese Projektarbeit ist noch nicht veröffentlicht worden. Sie ist somit weder anderen Interessierten zugänglich gemacht noch einer anderen Prüfungsbehörde vorgelegt worden.»

Name            Joshua Villing  
Ort            Aarau  
Datum        25.03.2022

Unterschrift .....