Fachhochschule Nordwestschweiz
Hochschule für Technik

Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem

IP6 - Bachelor Thesis

4. März 2022

Studenten	Joshua Villing
Fachbetreuer	Daniel Jossen
Auftraggeber	Daniel Jossen
Studiengang	Informatik
Hochschule	Hochschule für Technik

Management Summary

Lorem Ipsum

Inhaltsverzeichnis

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 Benachrichtigungen zu versenden und Sprachverbindungen zwischen Teilnehmern aufzubauen. Als Benutzeroberfläche dient dabei eine native iOS Applikation. Folgende Abbildungen zeigen die Startseite der iOS Applikation (Abbildung 1.1) sowie die Ansicht während eines aktiven Gruppenanrufs (Abbildung 1.2).

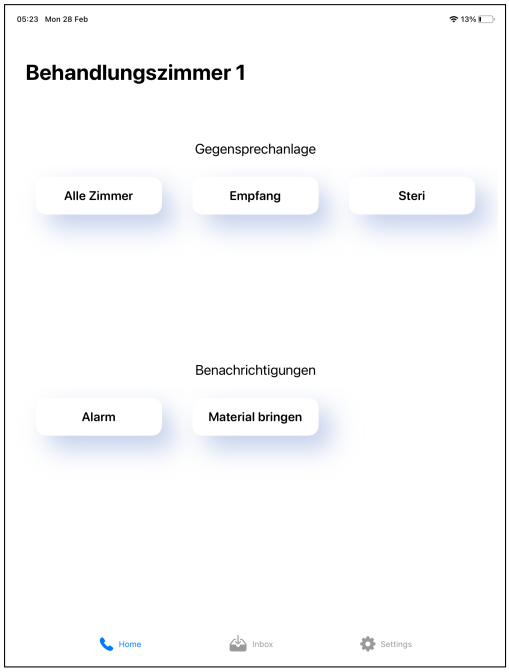


Abbildung 1.1: Praxisruf Startseite

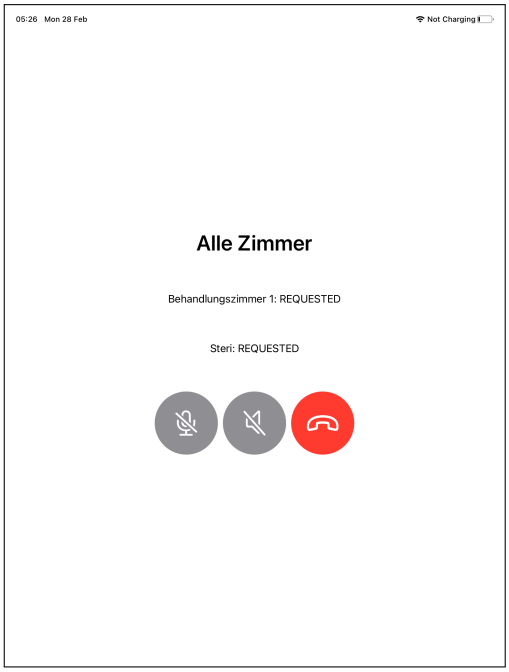


Abbildung 1.2: Aktiver Anruf

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 von empfangenen Benachrichtigungen automatisch 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. Für 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 bereits ein Praxisrufsystem mit eingeschränktem Funktionsumfang entwickelt. Die in diesem Projekt entwickelte Lösung ist eine Erweiterung des bestehenden Systems. Das zuvor umgesetzte System unterstützte das Versenden und Empfangen von Benachrichtigungen. Sprachsynthese für Benachrichtigungen und Sprachver-

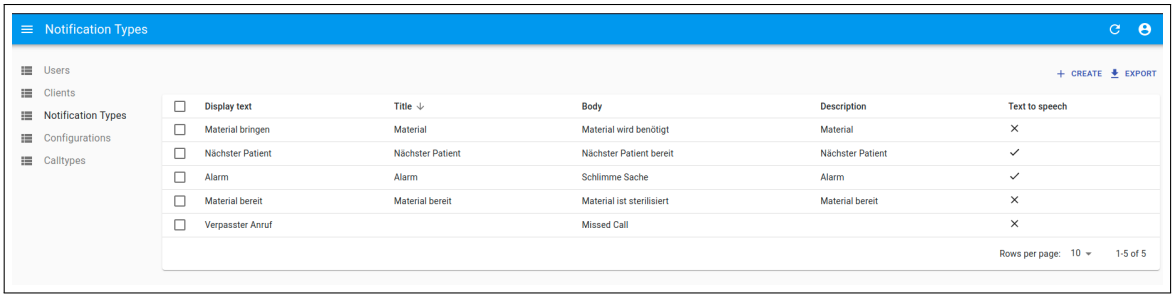


Abbildung 1.3: Praxisruf Startseite

bindungen für eine Gegensprechanlage konnten im Rahmen des Vorgängerprojektes nicht umgesetzt werden.[2] Die zentrale Aufgabenstellung für dieses Projekt war es, das System zu erweitern so, dass Benachrichtigungen vorgelesen und Sprachverbindungen aufgebaut werden können. Die Bedienung des Praxisrufsystems soll 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 unterstützten. 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 einsetzten. Diese Systeme sind 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 bereits teilweise. Mit dem 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 unbedingt als Gegensprechanlage verwendet werden können. Mit der Integration von Sprachsynthese für Benachrichtigungen kann sich das System weiter von bestehenden Lösungen absetzten. 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 könne effizienter Betrieb, Wartung sowie Hardware- und Betriebssystemkomaptibilität 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 eine 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 aus dem Cloudservice an Mobile Clients wird Firebase Cloud Messaging verwendet.

Die Synthese von Sprachdaten wurde mit einer Anbindung an AWS Polly im Cloudservice implementiert. Bei AWS Polly handelt es sich um eine Sprachsyntheseservice 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 implementierten. Dies bietet insbesondere die Vorteile, dass der Provider leicht ausgewechselt werden kann und, dass die Anbindung von zukünftige Plattformen übernommen werden kann.

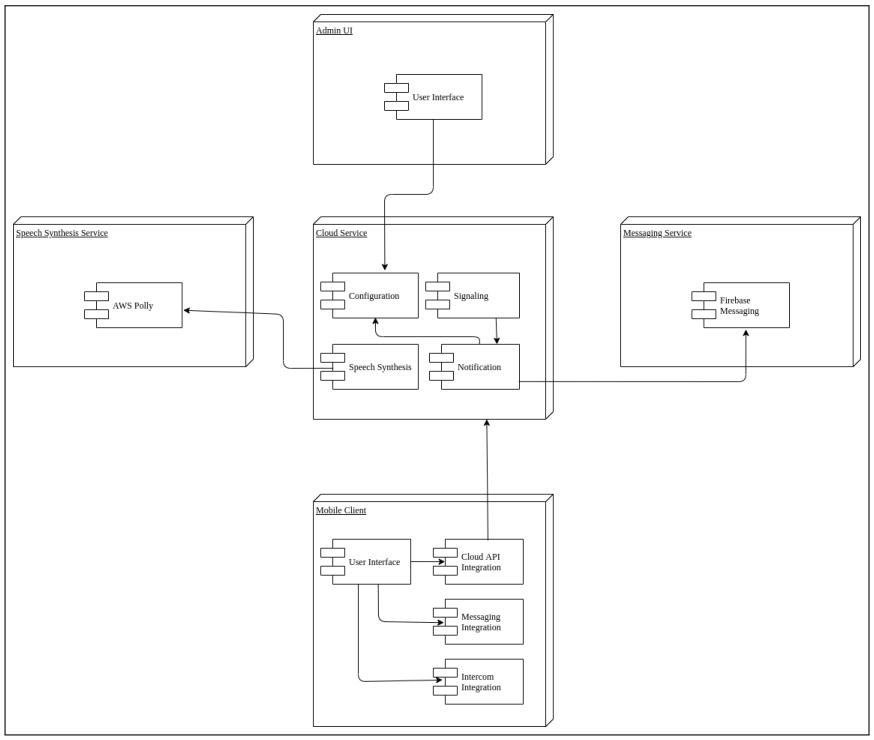


Abbildung 1.4: Systemarchitektur Praxisruf

Sprachverbindungen zwischen Mobile Clients werden als Peer To Peer Verbindungen aufgebaut. Diese Verbindungen wurden mit der Technologie WebRTC umgesetzt. Dabei handelt es sich um einen offenen Standard, welcher Echtzeitkommunikation für mobile Applikationen und Browser Applikationen ermöglicht.[4] Damit Peer To Peer Verbindungen zwischen Mobile Clients aufgebaut werden können, müssen diese synchrone Meldungen austauschen können. Um dies zu ermöglichen, wurde der Cloudservice um ein Modul ”Signaling” erweitert. Dieses Modul bietet eine Websocketschnittstelle, über welche die nötigen Signalmeldungen ausgetauscht werden können. Jeder Mobile Client der für Sprachverbindungen verfügbar ist, baut bei der Anmeldung eine Verbindung zum Signaling Modul auf. Über diese Verbindung sendet und empfängt er alle Signalmeldungen, die für den Verbindungsaufbau 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 folgt eine Technologie Evaluation für Sprachsynthese und Gegensprechanlage. Dabei werden mögliche Optionen beschrieben und anschliessend begründete Entscheidungen getroffen. Das darauffolgende Kapitel beschreibt das detaillierte technische Konzept für Funktionsweise und Architektur des Systems. Es wird beschrieben, wie die Funktionen Gegensprechanlage und Sprachsynthese umgesetzt wurden. 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.

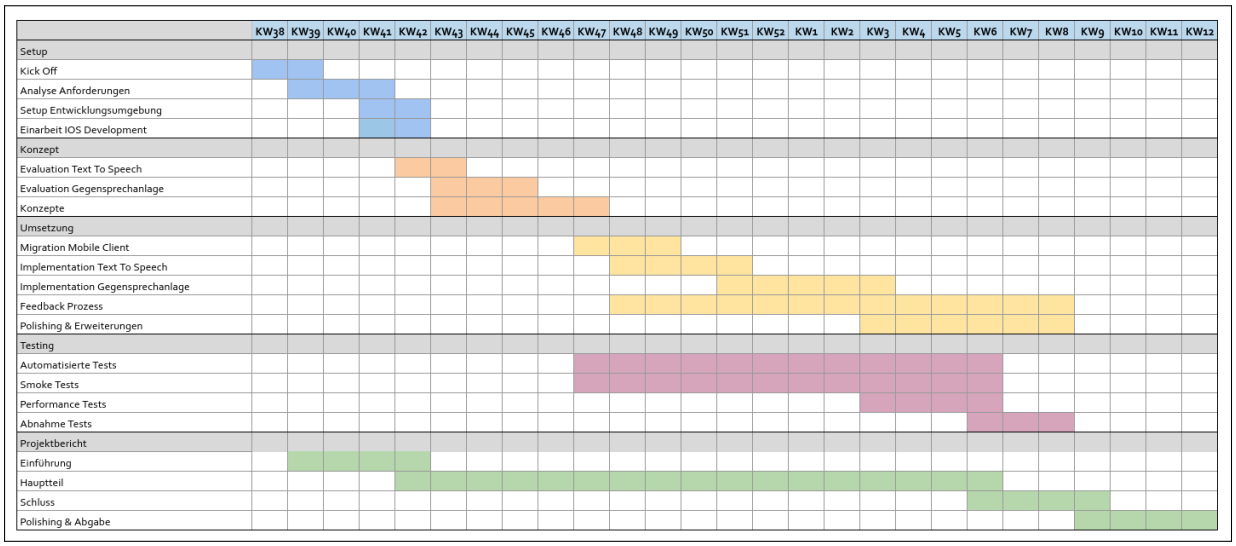


Abbildung 2.1: Projektplan

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 wird 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 integrieren. Dabei müssen insbesondere Skalierbarkeit, Erweiterbarkeit und Integrierbarkeit des Systems bewahrt werden. Die Umsetzung des Systems findet schliesslich in der dritten Phase. 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. Darauf werden Sprachsynthese und letztlich 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 ein Abnahmetest mit dem Kunden durchgeführt. Die vierte Phase bildet den Abschluss des Projekts. Es werden keine Änderungen mehr an der Applikation vorgenommen. Der Projektbericht wird fertiggestellt und das Projekt wird für die Abgabe vorbereitet.

2.2 Meilensteine

In der Anfangsphase des Projektes wurden folgende Meilensteine definiert:

Id	Beschreibung
M01	**Initiale Anforderungsanalyse** Die Anforderungen an das Projekt aus der Aufgabenstellungen sind in User Stories dokumentiert.
M02	**Einarbeit und Setup IOS Umgebung** Projektteilnehmende sind mit groben Konzepten der IOS Entwicklung vertraut. Die Entwicklungsumgebung ist bereit für die Umsetzung.
M03	**Evaluation Technologien** Die Evaluation der Technologien für Sprachsynthese und Gegensprechanlage ist abgeschlossen.
M04	**Konzepte** Die Konzepte für Systemarchitektur, mobile Applikation, Gegensprechanlage und Sprachsynthese sind erarbeitet.
M05	**Migration betehender Funktionalität** Die Funktionen die im Mobile Client der Projektarbeit "IP5 Cloudbasiertes Praxisrufsystem" umgesetzt wurden, stehen in der neuen nativen IOS Applikation zur Verfügung.
M06	**Umsetzung Sprachsynthese** Die Anforderung zur Funktion Sprachsynthese sind umgesetzt.
M07	**Umsetzung Gegensprechanlage 1:1** Die Anforderung zur Funktion Gegensprechanlage sind für 1:1 Verbindungen umgesetzt.
M08	**Umsetzung Gegensprechanlage 1:n** Die Anforderung zur Funktion Gegensprechanlage sind für 1:n Verbindungen umgesetzt.
M09	**Polishing und Erweiterungen** Die Applikation wurde eingehend getestet. Bekannte Fehler sind behoben oder dokumentiert. Gewünschte Anpassungen und Erweiterungen sind umgesetzt oder dokumentiert. Bei allen nicht umgesetzten Anpassungen ist beschrieben, wieso diese nicht umgesetzt werden konnten.
M10	**Abnahme** Die Abnahmetests wurden zusammen mit dem Kunden ausgeführt.
M11	**Projektbericht** Der Projektbericht ist vollständig.
M12	**Abgabe** Projektbericht und Quellcode sind fertiggestellt. Das Projekt ist abgegeben.

2.3 Abweichungen

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 die mangelhafte Dokumentation der verwendeten Bibliotheken zurückzuführen. Andererseits wurde schlicht zu wenig Puffer für die Implementation eingeplant. Die Verzögerungen während der Umsetzung hat dazu geführt, dass weniger Zeit als erhofft für Polishing und Erweiterungen aufgewendet werden konnten.

Die in Kapitel 2.2 definierten Meilensteine sind grösstenteils erreicht. Der Milestone M09 Polishing und Erweiterungen wurde nicht vollständig erreicht. Wegen Verzögerungen in der Umsetzungsphase konnte weniger Zeit als erwartet für Polishing und Erweiterungen aufgewendet werden. Es wurden deshalb keine Erweiterungen die über die Mindestanforderungen herausgehen umgesetzt.

3 Anforderungen

Dieses Kapitel beschreibt die Anforderungen an das Praxisrufsystem, welche zu Beginn des Projektes erarbeitet werden. Die Anforderungen werden aus fachlicher der Stakeholder festgehalten. Dazu werden zunächst Stakeholdergruppen identifiziert und anschliessend pro Gruppe User Stories definiert. Jede User Story beschreibt eine Anfroderung welche eine Stakeholdergruppe an das System hat. Anhand der User Stories werden anschliessen 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 drei Stakeholdergruppen, welche für dieses Projekt übernommen werden. Die Gruppe Praxismitarbeitende verwendet das Praxisrufsystem, um in der Praxis zu kommunizieren. Neben Praxismitarbeitenden, arbeitet auch die Rolle des Praxisverantwortlichen mit dem Praxisrufsystem. Praxisverantwortliche sind dafür verantwortlich, die Konfiguration des Praxisrufsystems vorzunehmen. Als dritte Stakeholdergruppe hat der Auftraggeber ein Interesse daran, dass gewisse Rahmenbedingungen gesetzt und eingehalten werden.[2] Im Folgenden werden die Anforderungen pro Stakeholdergruppe tabellarisch aufgelistet.

3.1.1 Praxismitarbeitende

Id	Anforderung
U01	Als Praxismitarbeiter/in möchte ich alle Funktionen aus der existierenden Applikation weiterhin verwenden können, damit mir diese weiterhin die Arbeit erleichtern. [2]
U02	Als Praxismitarbeiter/in möchte ich, dass eingehende Benachrichtigungen vorgelesen werden, damit 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 es die Funktion einer Gegensprechanlage wirklich erfüllt.
U07	Als Praxismitarbeiter/in möchte ich nur Buttons für Sprachverbindungen sehen, die für mich relevant sind.
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 Praxisadministrator

Id	Anforderung
U13	Als Praxisverantwortlicher möchte ich konfigurieren können, welche Benachrichtigungen dem Praxismitarbeitenden vorgelesen werden, damit nur relevante Benachrichtigungen vorgelesen werden.
U14	Als Praxisverantwortlicher möchte ich konfigurieren können, aus welchen Zimmern Sprachverbindungen zu welchen anderen Zimmern aufgebaut werden können, damit die Mitarbeitendend das System effizient bedienen können.
U15	Als Praxisverantwortlicher möchte ich Benachrichtigungen, Clients und Benutzer wie zuvor konfigurieren können, damit ich das System weiterhin auf meine Praxis zuschneiden und bestehende Konfigurationen übernehmen kann.

3.1.3 Auftraggeber

Id	Anforderung
U16	Als Auftraggeber möchte ich die 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, der bestehende Mobile Client als native iOS Applikation ungeschrieben wird, um Wartbarkeit und Gerätekompatibilität langfristig zu gewährleisten.
U19	Als Auftraggeber möchte ich, dass wo möglich der Betrieb von Serverseitigen Dienstleistungen über Amazon Webservices betrieben wird, damit ich von bestehender Infrastruktur und Erfahrung profitieren kann.

3.2 Features

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

Id	Feature
F01	Migration des bestehenden Mobile Client
F02	Sprachsynthese
F03	Gegensprechanlage

Das Feature F01 Migration des bestehenden Mobile Client bildet die Grundalge für die native iOS Applikation. Damit wird die native iOS Applikation mit dem Funktionsumfang der Vorgängerversion behandelt. Das Feature Sprachsynthese baut auf F01 auf und fügt die Sprachsynthese für empfangene Benachrichtigungen hinzu. Mit dem Feature Gegensprechanlage wird schliesslich das Kernstück der neuen Funktionalität behandelt. Dieses Feature behandelt 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, Testschritt und die erwarteten Resultate definieren. Die Szenarien sind im Anhang D aufgeführt. Die Resultate der letzten Ausführung der Testszenarien sind im Kapitel 6 festgehalten.

4 Ausgangslage

Dieses Kapitel beschreibt das Umfeld in dem diese Projektarbeit ausgeführt wird. Dabei wird eine grobe Übersicht zu der im Vorgängerprojekt erarbeiteten Lösung gegeben. Es wird weiter ein Überblick über vergleichbare bestehende Systeme gegeben.

4.1 Bestehende Rufsysteme

Im Vorfeld dieses Projektes wurde eine Marktanalyse durchgeführt 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. Weiter seien bestehende Systeme weder über TCP/IP Netzwerke integrierbar, noch über externe eine API ansteuerbar.[1]

Durch das Projekt ”IP5 Cloudbasiertes Praxisrufsystem” wurde ein im Funktionsumfang eingeschränktes cloudbasiertes Praxisrufsystem entwickelt. Dieses wird im folgenden Kapitel vorgestellt.

4.2 Vorarbeit Cloudbasiertes Praxisrufsystem

Das im Projekt ”IP5 Cloudbasiertes Praxisrufsystem” entwickelte Praxisrufsystem ermöglicht es vorkonfigurierte Benachrichtigungen zwischen Endgeräten zu versenden. Sprachverbindungen und Sprachsynthese für Benachrichtigungen werden von diesem System nicht unterstützt.

Das bestehende System umfasst eine zentralen Serverkomponente, eine Weboberfläche und eine mobile Applikation. Die mobile Applikation dient als Endgeräte 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 REST Api an, über welche Konfigurationen erfasst und gelesen werden können. Diese Api wird von der Weboberfläche ”Admin UI” angesprochen, um Konfigurationen zu erstellen und bearbeiten. Sie wird weiter von der mobilen Applikation verwendet, um die Konfiguration der Applikation zu laden.

Das Senden und Empfangen von Benachrichtigungen im bestehenden System wird durch Firebase Cloud Messaging (Firebase Cloud Messaging) ermöglicht. Sowohl in der Mobile Client als auch im Cloudservice ist eine Anbindung an FCM implementiert. Dabei wird die Anbindung auf der Seite des Mobile Clients ausschliesslich zum Empfangen von Meldungen verwendet. Das Versenden von Benachrichtigungen wird an den Cloudservice delegiert. Dazu sendet ein Mobile Client eine Anfrage an die REST Api des Cloudservice. Dieser wertet die Konfiguration aus und benutzt die FCM Anbindung um eine Benachrichtigung an alle relevanten Empfänger zuzustellen.

Sämtliche Infrastruktur für das bestehende Praxisrufsystem ist mit Amazon Webservices (AWS) eingerichtet. Der Cloudservice besteht aus einer einzelnen Java-Applikation. Diese wird mit einer AWS Elastic Beanstalk Instanz betrieben. Das Admin UI ist eine simple Javascript Applikation. Sie wird mit AWS Amplify betrieben.[2]

5 Technologieentscheid

In diesem Kapitel wird entschieden, mit welchen Technologien und Frameworks die Anforderungen für 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 IOS App

Mit dem Projekt "IP5 Cloudbasiertes Praxisrufsystem" eine mobile Applikation entwickelt, die als Endpunkt in einem Praxisrufsystem verwendet werden kann. Diese unterstützt das Senden und Empfangen vorkonfigurierter Benachrichtungen [2]. Die Applikation wurde auf Basis des Frameworks Nativescript [5] als Multi-Platform Applikation entwickelt. Dadurch ist es möglich dieselbe Codebasis für die Entwicklung von Android und iOS Applikationen zu verwenden. Im Fazit des Vorgängerprojekts wurde empfohlen, diese Applikation durch native Applikationen pro Platform zu ersetzten. Dadurch könne 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 Benachrichtungen senden und empfangen zu können, muss die gewählte Technologie es ermöglichen Firebase Cloud Messaging anzubinden und Push Benachrichtigungen im Vorder- sowie im Hintergrund empfängen können. Weiter muss die Technologie es ermöglichen, regelmässige Aufaben auszuführen. Dadurch kann überprüft werden, ob der Benutzer ungelesene Benachrichtigungen hat und wenn nötig einen 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: SSwiftUI 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 declarative 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 umzusetzten.

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 aufgehoben, 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 umuzusetzen.

5.1.3 Unterstützung Benachrichtigungen

Die neue iOS Applikation, muss es weiterhin ermöglichen Benachrichtigungen über Firebase Cloud Messaging zu empfangen und versenden. Firebase stellt dazu eine native iOS Library 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 sich in den Lifecycle des Betriebssystems einzuhängen [12]. Dadurch ist es auch möglich, Vorder- und Hintergrundbenachrichtigungen über das Benachrichtigungszenter von iOS anzuzeigen [11].

Die Firebase Cloud Messaging Library kann sowohl mit SwiftUI als auch mit UIKit verwendet werden. AppDelegates sind ein Konzept welches aus UIKit stammt[12]. SwiftUI Applikationen können ohne AppDelegates implementiert werden. UIKit Funktionen können allerdings nahtlos mit SwiftUI integriert werden.[9] Die Firebase Cloud Messaging Library für iOS ermöglicht es also, Benachrichtigungen von Praxisruf sowohl mit UIKit als auch mit SwiftUI umzusetzten.

Um Konfigurationen zu laden und Benachrichtigungen zu versenden, muss die REST Api des Cloudservice angesprochen werden können. Dies ist über die URLRequest-Komponente aus der iOS Standardbibliothek möglich [13].

5.1.4 Benachrichtigungen prüfen

Die mobile Applikation aus dem Vorgängerprojekt erinnert in regelmässigen Abständen, wenn ungelesene Benachrichtigungen vorhanden sind. Diese Funktion muss auch von der neuen iOS Applikation unterstützt werden. Dazu muss regelmässig geprüft werden, ob es ungelesene Benachrichtigungen gibt und gegebenenfalls ein Erinnerungston abgespielt werden. Die standard iOS Bibliothek bietet Mittel, mit welchen regelmässige Aufgaben angestossen werden können. Einerseits können über eine ”Timer”-Komponente in Regelmässigen abständen Events veröffentlicht werden. Auf einer SwiftUI View kann ein beliebiger Listener registriert werden, der beim Empfang eines Events des Timers aufgerufen wird. 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, könnte mit einem Timer trotzdem genau dasselbe Verhalten wie in der bestehenden Applikation umgesetzt werden. Die Standardbibliothek bietet allerdings auch Mittel, um Aufgaben im Hintergrund zu verarbeiten. Über die Klasse BGTaskScheduler können Aufgaben erfasst werden, die im Hintergrund ausgeführt werden [15].

Die Erinnerungsfunktion kann mit Mitteln aus der iOS Standardbibliothek umgesetzt werden.

5.1.5 Entscheid

Der native iOS Mobile Client für Praxisruf wird mit Swift und SwiftUI umgesetzt. Die deklarative Syntax von SwiftUI, ermöglicht es einfacher übersichtliche und lesbare Komponenten zu implementieren. Integration in die XCode Entwicklungsumgebung und verfügbare Standardkomponenten vereinfachen die Entwicklung. Es ist möglich, dass einige Funktionen noch nicht mit SwiftUI umgesetzt werden können, weil dafür benötigte Features 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 SwitUI verfügbar sind, migriert werden.

Als Zielplattform für die Applikation wird iOS15 verwendet. Damit wird die neuste iOS Version unterstützt. Dies ermöglicht es, alle verfügbaren SwiftUI Features zu verwenden und minimiert die Wahrscheinlichkeit, auf UIKIt zurückgreifen zu müssen.

5.2 Sprachsynthese

Das Feature Sprachsynthese fordert, dass das System das Vorlesen empfangener Benachrichtigungen unterstützt. Um dies zu ermöglichen muss eine Technologie integriert werden, die es erlaubt aus Sprachdaten aus Textinhalten zu synthetisieren. Diese Sprachdaten müssen als Audiodateien vom Mobile Client abgespielt werden können.

Diese Integration kann mit den Standardbibliotheken für iOS oder durch die Anbindung eines externen Providers umgesetzt werden. Die Anbindung eines externen Providers kann direkt im Mobile Client implementiert werden. Alternativ kann die Serverkomponente Cloudservice an den Provider 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

Vom Auftraggeber ist explizit gewünscht, dass für Infrastruktur und Dienstleistungen wo möglich Amazon Webservices verwendet wird[1]. Mit dem Amazon Polly bietet Amazon Webservices einen Service, welcher Text in Sprachdaten verwandeln kann [3]. Amazon Webservices stellt Libraries für iOS [17], Android [18] und für Java zur Verfügung [19]. 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 Providers bleibt auch in dieser Variante aufwändig. Weiter bringt auch diese Variante den Nachteil, dass der Mobile Client komplexer wird, weil er mit einer zusätzlichen Instanz kommunizieren muss.

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 auszutauschen. Gleichzeitig hat diese Variante den Nachteil, dass der Cloudservice komplexer wird.

[1] Siehe Kapitel 3.1.3

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 Schnittstelle der anderen Module bezieht.

Als dritte Option für die Integration von Amazon Polly kan AWS Lambda verwendet werden [3]. AWS Lambda erlaubt es Funktionalität serverless auszuführen. Der entsprechende Code läuft in diesem Fall nicht als klassischer Server, sondern wird nur bei Bedarf ausgeführt [20] Die Integration von Amazon Polly über AWS Lambda kann für das Praxisrufsystem 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 müssen die persistierten Sprachdaten geladen und zurückgegeben werden. Die Verwendung von AWS Lambda hat damit den Nachteil, dass die synthetisierten Sprachdaten zwingend ausserhalb des Mobile Clients persistiert werden müssen. Weiter findet damit eine zusätzliche Bindung an Amazon statt. Gleichzeitig bietet es den Vorteil, dass weniger Infrastruktur benötigt wird, da die Abfrage an AWS Polly ohne dedizierten Server abgesetzt werden kann.

5.2.3 Entscheidung

Die Sprachsynthese wird durch die Anbindung des externen Providers 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 zu persistieren.

Durch diesen Ansatz wird die Abhängigkeit zu einzelnen Provider 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 Sprachübertragung

Die zentrale Aufgabenstellung dieser Arbeit dreht sich darum, Peer-To-Peer Sprachübertragung in ein cloudbasiertes Praxisrufsystem zu integrieren. Sprachübertragung in Echtzeit muss in Form von Anrufen zwischen Clients ermöglicht werden. Diese Erweiterung ermöglicht es, das Praxisrufsystem als Gegensprechanlage zu verwenden. Dieses Kapitel beschreibt zwei Ansätze, mit welchen diese Anforderung umgesetzt werden kann. Der erste Ansatz beinhaltet die Anbindung eines bestehenden Telefonieproviders. Der zweite Ansatz verzichtet auf die Integration eines Providers und setzt Sprachverbindungen als direkte Peer-To-Peer Verbindungen zwischen Clients 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[2]. 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].

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 und die Abläufe, die Amazon Chime zum Verbindungsaufbau vorsieht, stattfindet. Dabei 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. Um den Austausch diese 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

[2] Siehe Kapitel 3.1.3

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. Der Signaling Service 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 können über die bereits implementierte 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 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 des Praxisrufsystems kann das Problem dadurch adressiert werden, dass nicht erreichbare Endgeräte durch asynchrone 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 [23]. 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 [24].

5.3.3 Entscheidung

Sprachübertragung wird 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. Verfügbarkeit der Signaling Instanz wird den Betrieb des Cloudservices sichergestellt.

6 Peer-To-Peer Sprachübertragung

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

6.1 Kommunikationsprotokolle

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 Sprache oder Video übertragen werden [25]. Diese Beschreibung beinhaltet Metainformationen zu einer Verbindung. Dazu gehören unter anderem Auflösung, Format sowie Verschlüsselung und Codierung der zu übertragenden Daten [26]. WebRTC verwendet SDP um zu Beschreiben, welche Art von Daten mit einer Verbindung übertragen werden und wie diese verarbeitet werden können [27]. 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 Verbindung Kommunikation stattfinden kann [28]. ICE wird in WebRTC verwendet, um die Netzwerkverbindung zwischen zwei Teilnehmern aufzubauen [26]. Der Austausch von ICE Candidates bildet den zweiten Teil an Signalmeldungen, welche über die Signaling Instanz ausgetauscht werden müssen [27].

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 [27].

Die WebRTC Bibliothek für iOS implementiert dieses Interface. Es bietet die Möglichkeit eine RTCPeerConnection 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. Dieses kann darauf die RTCPeerConnection auf seiner Seite initialisieren und entsprechende SDP Informationen über die Signaling Instanz zurücksenden.

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. 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 on 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 Signale entgegennehmen und an relevante Empfänger weiterleiten können.

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. Konzept und Funktionsweise der Signaling Instanz für das Praxisrufsystem wird in Kapitel 7 beschrieben.

6.4 Unicast, Multi-Cast, Broadcast

Dieses Kapitel beschreibt, der Verbindungsaufbau von Peer-To-Peer Verbindungen mit WebRTC als Uni-, Multi- und Broadcast in einem cloudbasierten Praxisrufsystem umgesetzt werden kann.

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. Dies bedeutet, dass der Verbindungsaufbau wie in Kapitel 6.2 beschrieben für jedes Ziel einzeln ausgeführt werden muss.

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. Jeder der Empfänger kann die Signale gleich wie in einer Unicast Verbindung verarbeiten. 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.

Um eine n:n Verbindung aufzubauen und die Kommunikation zwischen allen Teilnehmern zu erlauben, müsste jeder Teilnehmer mit jedem anderen Teilnehmer eine Verbindung aufbauen [23]. Im Rahmen dieser Arbeit sind lediglich 1:n Verbindungen gefordert. Konzepte und Umsetzung für n:n Verbindungen werden hier deshalb nicht weiter behandelt.

6.5 Sicherheit

Signaling Verbindung muss sicher sein. Keine Anforderung an Protokoll, ausser das verschlüsselt.

Übertragung der Sprachdaten muss verschlüsselt sein. Für die Verschlüsselung muss SRTP verwendet werden. Die Verschlüsselung ist Sache von dem der das Interface implementiert. Ist dementsprechend in der Library mit implementiert.

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. Die folgende Dokumentation beschreiben die neuste Version des Konzepts und stellen 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. Nach dem Systemkonzept werden Konzepte für die Umsetzung der drei Features ”Migration des bestehenden Mobile Client”, ”Sprachsynthese” und ”Gegensprechanlage” beschrieben. Das Kapitel schliesst mit einem Überblick über die Domänen- und Servicemodelle sowie der Schnittstellen welche die API des Systems bietet.

7.1 Systemarchitektur

Dieses Kapitel beschreibt die Komponenten des Praxisrufsystems und wie diese erweitert werden. Das System wird um Komponenten zur Signalvermittlung und Sprachsynthese erweitert. Weiter wird die interne Struktur des Cloudservices angepasst um die Weiterentwicklung und Betrieb des Systems zu vereinfachen.

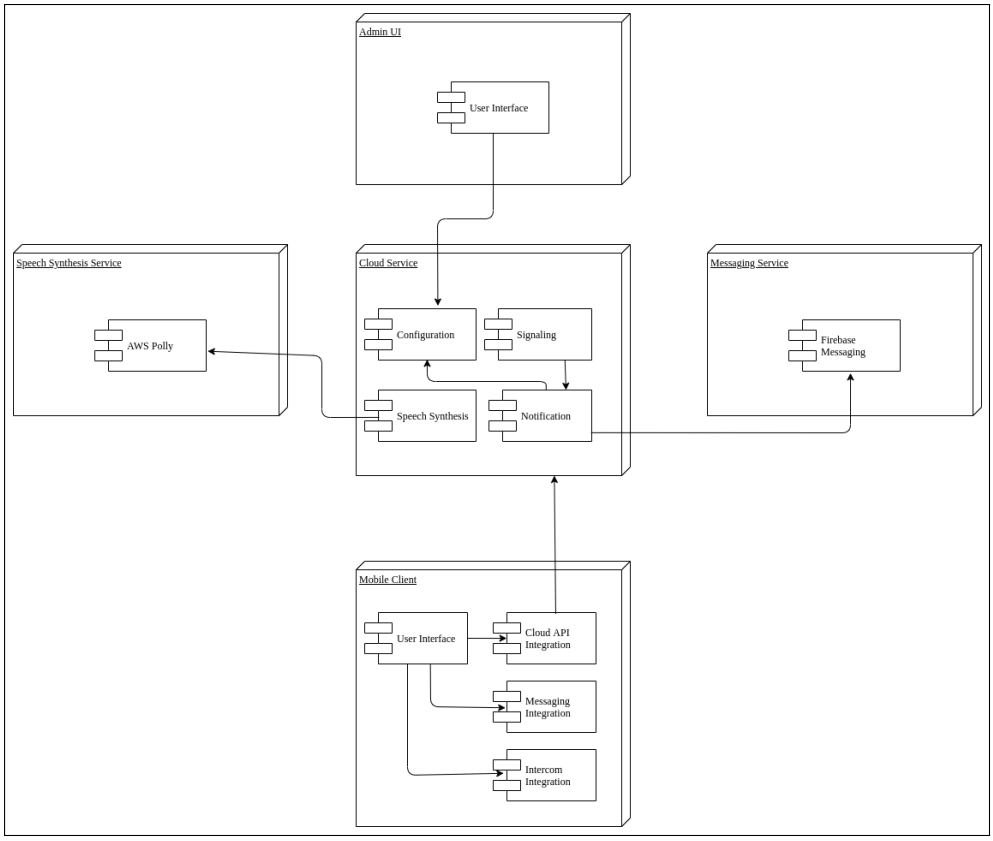


Abbildung 7.1: Systemkomponenten

Abbildung 7.1 gibt einen Überblick über die Systemkomponenten und welche Teile diese beinhalten. Die Pfeile zwischen Komponenten zeigen gerichtete Abfragen und damit eine Funktionale Abhängigkeit. Alle dargestellten Komponenten sind entkoppelt und Abfragen finden nur über definierte Schnittstellen statt.

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 das zentrale Serverkomponente des Praxisrufsystems. 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 Notification Domain eine Abfrage an die Configuration Domain geschehen. Die Configuration Domäne bietet dazu eine REST 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 Module aufgeteilt. Diese Module werden weiterhin in einer einzelnen Applikation zusammengefasst. Die Modularisierung können aber zu einem späteren Zeitpunkt in einzelne Microservices aufgeteilt werden.

Nachdem die Auftrennung in Module stattgefunden hat, wird der Cloudservice um die zwei Module Signaling und Speech Synthesis erweitert. Das neue Modul Signaling ist für die Signalvermittlung zwischen Mobile Clients verantwortlich. Es übernimmt die Aufgabe der Signaling Instanz für den Aufbau von Peer-To-Peer Sprachverbindungen mit WebRTC. Das Modul Signaling hat eine gerichtete Abhängigkeit zum Module Notification. Über das Notification Modul sollen Empfänger informiert werden, wenn ein Signal nicht zugestellt werden konnte. 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 Appliaktion, über welche das Praxisrufsystem 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. Die native iOS App wird von Grund auf neu entwickelt. Dabei muss sämtliche Funktionen des bestehenden Mobile Clients als Teil der nativen App neu implementiert werden. 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 noch nicht vorgenommen. Die beiden Domänen wurden lediglich durch die Package Struktur innerhalb eines einzelnen monolithischen Services getrennt.

Die Trennung der Domänen innerhalb des Cloudservice wird weiter verstärkt. Teile der Applikation werden neu in 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 REST-Schnittstellen geschehen.

Es werden die vier Domänen-Module Configuration, Notification, Spech 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 Benachrichtiungen. Das Modul Speech Synthesis beinhaltet die Anbindung an den Speech Synthesis Service und stellt eine Schnittstelle zur verfügung über den das restliche System Sprachdaten beziehen kann. Das Modul Signaling 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 für Security und Fehlerhandling. Neben den Modulen App und Commons, werden vier weitere Module für die Domänen Configuration, Notification, Speech Synthesis und Signaling erstellt.

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.

7.2 Nativer Mobile Client

Dieses Kapitel beschreibt das Konzept für einen nativen iOS Client zur Bedienung des cloudbasierten Praxisrufsystems. 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 sowie Anbindung von Firebase Cloud Messaging. Die Integration von Gegensprechanlage und Sprachsynthese wird nur im Rahmen des Entwurfs der Benutzeroberfläche erwähnt. Eine detaillierte Beschreibung dieser Konzepte 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 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 7.2: Mockup Login

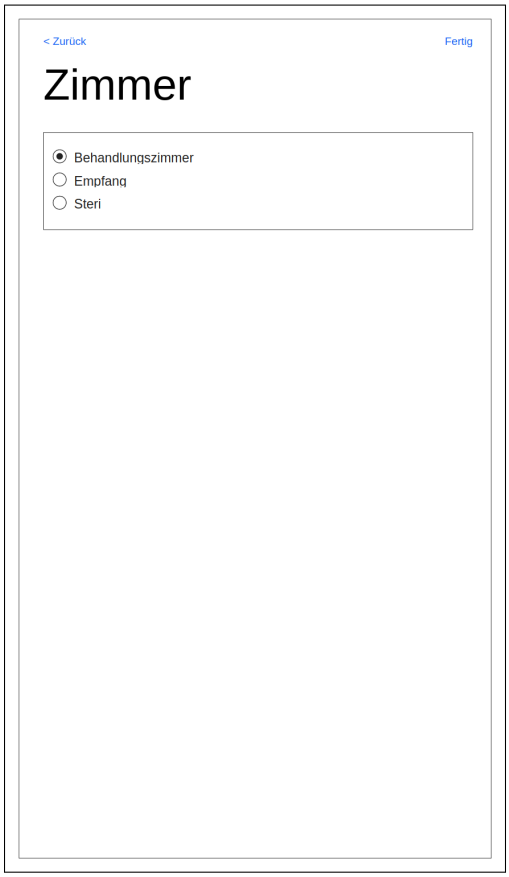


Abbildung 7.3: Mockup Zimmerwahl

Nach dem Eingeben der Anmeldedaten, erhält der Benutzer die Möglichkeit die gewünschte Konfiguration auszuwählen. Die Ansicht 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 Logindaten. 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. 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 entsprechenden 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 über der Anruf über einen roten Button am rechten Rand beendet werden. Nach einem beendeten Anruf wird automatisch zu der zuvor angezeigten Ansicht navigiert.

Abbildung 7.4: Mockup Home

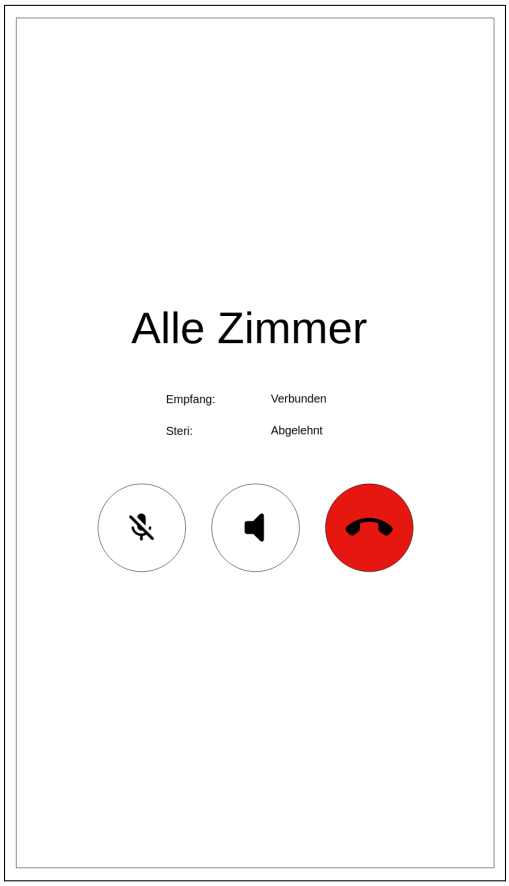


Abbildung 7.5: Mockup Aktiver Anruf

Der Bereich Inbox zeigt eine Liste der empfangenen Benachrichtigungen sowie der empfangenen und verpassten Anrufe. Für Benachrichtigungen wird der Titel der Benachrichtigung gefolgt vom Namen des Versenders in Klammern sowie der Inhalt der Benachrichtigungen angezeigt. Für Anrufe wird der Name des Anrufers angezeigt. Zudem wird 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 gesehen werden. Dazu wird im Abstand von 60 Sekunden geprüft, ob unquittierte Benachrichtigungen oder Anrufe vorhanden sind. Ist dies der Fall, wird ein Erinnerungston abgespielt und eine Benachrichtigung angezeigt. Dieser Mechanismus wird in Kapitel 7.2.4 weiter beschrieben.

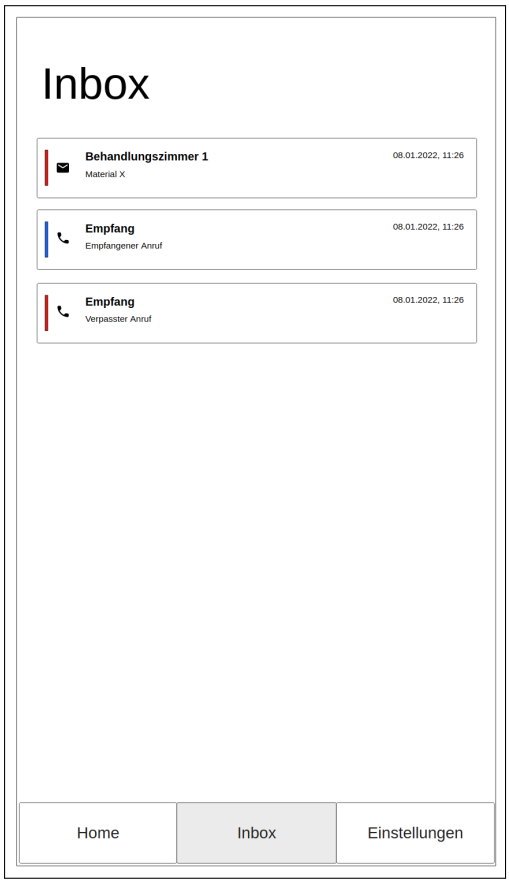


Abbildung 7.6: Mockup Inbox

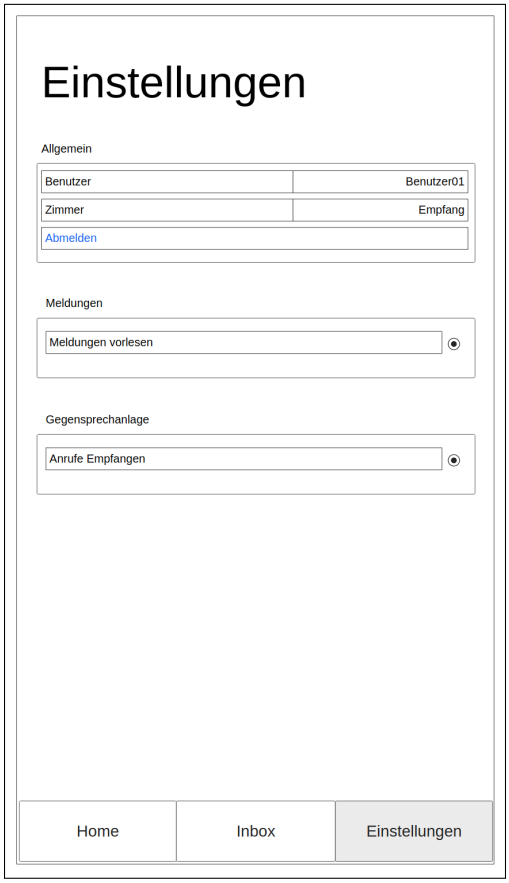


Abbildung 7.7: Mockup Einstellungen

Abbildung 7.7 zeigt den Bereich Einstellungen. Der Bereich Einstellungen zeigt den aktuellen Benutzernamen und die gewählte Konfiguration. Über die Schaltfläche Abmelden, können sich Praxismitarbeitende 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

Der Mobile Client muss an die API des Cloudservices angebunden werden. Es wird eine Anbindung an die Domäne Configuration zur Anmeldung und Auswahl des gewünschten Zimmers, an die Domäne Notification zum Versenden von Benachrichtigungen und an die Domäne Speech Synthesis für den Bezug von Sprachdaten benötigt. Die Schnittstellen dieser Domänen stehen als HTTP Endpunkte zur verfügung. In diesem Unterkapitel wird beschrieben, wie REST Aufrufe 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 Netzwerkaufrufe getätigt werden können. Über URLSession.shared steht eine Standard-Instanz zur Verfügung, über

welche Netzwerkanfragen verarbeitet werden können [29]. Die Klasse UrlRequest ermöglicht es, Http-Request 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 Absetzten der nötigen UrlRequest Instanzen. Er bietet öffentliche für die Http Verben Get, Post und Delete an über welche entsprechende Http-Requests abgesetzt werden können. Das Klassendiagramm in Abbildung 7.8 zeigt den Aufbau des Service PraxisrufApi.

Die Basis URL für API Anfragen wird in der Konfiguration der iOS Applikation definiert. Der PraxisrufApi Service lädt diese und verwendet sie als Basis 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 und Put von PraxisrufApi nehmen zudem einen optionalen Parameter von Typ Data entgegen. Dieser definiert den Inhalt des Request Bodies. Mit diesen Informationen kann der Http Request erstellt und versendet werden.

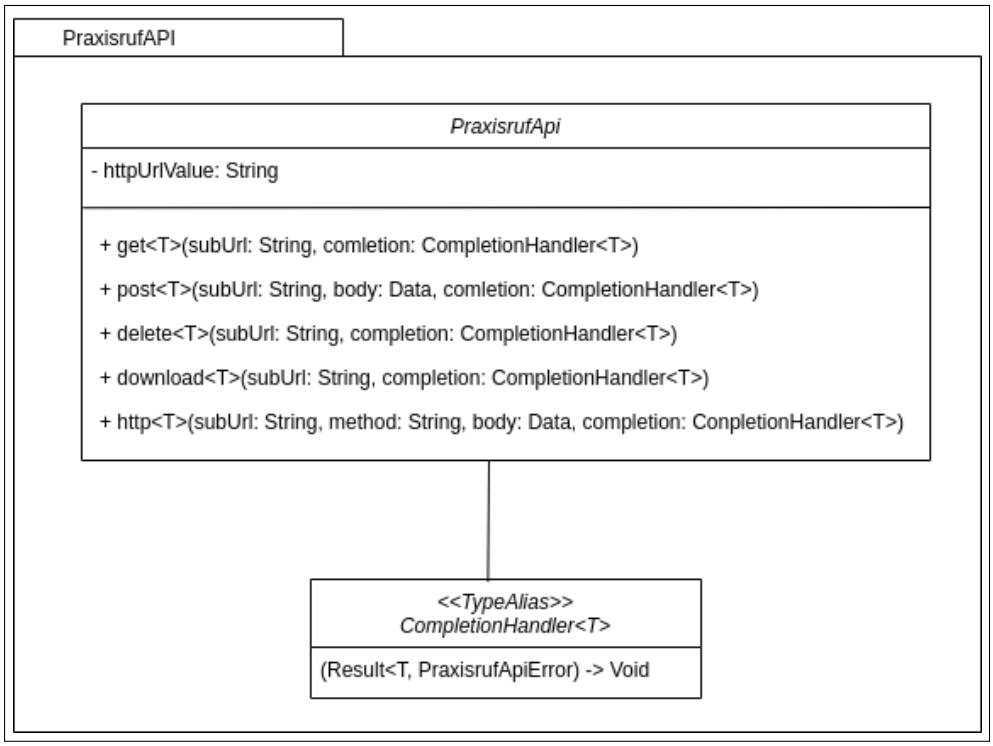


Abbildung 7.8: Klassendiagramm PraxisrufApi

Sämtliche öffentlichen Methoden des PraxisrufApi Service nehmen einen Parameter mit dem Namen completion entgegen. Dabei handelt es sich um ein Callback, welches beim Erfolg oder Fehlschlagen der Http Anfragen ausgeführt wird. Als Input dieses Callbacks 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 [30]. Im Fehlerfall wird das Result mit einem PraxisrufApiError Objekt erstellt. Im Erfolgsfall wird der Wrapper mit einem Object vom Typ T erstellt. Der Typ T ist generisch und wird vom Aufrufer der PraxisrufApi definiert. Der Typ kann grundsätzlich beliebig sein, er muss aber das Protokoll Decodable aus der iOS Standardbibliothek unterstützen. Decodable Instanzen können von einer JSON Spring Repräsentation in ein Swift Object konvertiert werden [31]. So kann das Resultat im Erfolgsfall aus der Response generiert und an, dass Callback übergeben werden. Dadurch kann die Konvertierung generisch in der Basisklasse 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 des completion-Callbacks statt. Dieser Ansatz ermöglicht es im PraxisrufApi-Service ausschliesslich den API Call abzu-

handeln. Der Api Service muss keinen State führen und kann generisch für alle Anwendungen wiederverwendet werden.

Requests die über den Api Service erstellt werden, werden automatisch authentisiert. Dazu lädt der Service die hinterlegten Credentials aus dem KeyStore von iOS und generiert einen entsprechenden Authorization Header. Ist kein Token vorhanden, wird keine Anfrage abgesetzt. Es wird direkt das completion-Callback mit einem Fehler-Resultat aufgerufen.

Mit dieser Lösung steht ein Service zur verfügung, über welchen Http Abfragen einfach integriert werden können. Dank der generischen Methoden im Basis Service können neue Calls einfach hinzugefügt werden, ohne das Boilerplate-Code wiederholt werden muss. Durch die completion Methoden kann zudem jede Abfrage den Bedürfnissen des Aufrufers entsprechend verarbeitet werden.

Um die Verwendung der API von Praxisruf weiter zu vereinfachen, wird der PraxisrufApi Service um sprechende Methoden für die nötigen Abfragen erweitert. Dazu wird pro Domäne eine Extension-Klasse definiert erstellt. Diese fügt Methoden mit sprechenden Namen für die angesprochene Funktionalität hinzu und kapseln die Verwendung der get, post und delete Methoden.

Der PraxsrufApi-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 API 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 der Messaging Service 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 Betriebssystem notwendig ist. Diese Integration kann bis heute nur über AppDelegates umgesetzt werden. SwiftUI Applikationen können oft ohne AppDelegates implementiert werden. Sobald aber Integration mit dem Betriebssystem notwendig ist, müssen AppDelegates verwendet werden. Dazu können AppDelegates bei der Initialisierung der Applikation registriert werden.

Zur Anbindung von Firebase Cloud Messaging wird dementsprechend ein AppDelegate implementiert. Die Logik des AppDelegates wird dabei auf das minimal Nötige reduziert. Der AppDelegate selbst ist für die direkte Kommunikation mit Messaging Service und Betriebssystem verantwortlich. Alle weitere Logik wird nicht im AppDelegate selbst ausgeführt, sondern an die Applikation delegiert. Dies ermöglicht es die Anbindung des Messaging Service im AppDelegate zu kapseln. Wird in Zukunft auf einen anderen Messaging Service gewchselt muss damit ausschliesslich die Logik im AppDelegate angepasst werden. Weiter sorgt diese Trennung dafür, 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.

Mit dem AppDelegate werden im Detail folgende Funktionen umgesetzt. 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. Der AppDelegate muss weiter die Möglichkeit bieten, den Client beim Messaging Service abzumelden. Für

die Verarbeitung von Benachrichtigungen muss der AppDelegate Benachrichtigungen im Vordergrund empfangen und als Push-Benachrichtigungen anzeigen können. Die Informationen aus der empfangenen Benachrichtigung müssen anschliessend an die Applikation übergeben werden. Der AppDelegate muss weiter Benachrichtigungen im Hintergrund empfangen und als Push-Benachrichtigung anzeigen können. Sobald die Applikation wieder in den Vordergrund tritt, müssen die Daten dieser Benachrichtigung an die Applikation zur weiteren Verarbeitung übergeben werden.

7.2.4 Scheduled Reminder für Inbox

Der bestehende Mobile Client prüft in regelmässigen Abständen, ob ungelesene Benachrichtigungen in der Inbox vorhanden sind. Wenn ungelesene Benachrichtigungen gefunden werden, wird ein Benachrichtigungston abgespielt, um den Praxismitarbeitende darauf aufmerksam zu machen. Diese Prüfung findet bisher nur statt, wenn die Applikation in Vordergrund aktiv ist und nicht, wenn sie minimiert ist. Diese Funktion wird mit dem nativen iOS Applikation übernommen. Dazu werden zwei Komponenten umgesetzt. Erstens wird InboxReminderService erstellt, welcher eine Liste der aktuellen Benachrichtigungen führt. Zweitens benötigt es eine Komponente, welche diesen InboxService regelmässig überprüft und bei Bedarf den Erinnerungston abspielt. Die Standardbibliothek für iOS bietet eine Timer Klasse. 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

Es muss eine sichere Übertragung von Daten gewährleistet sein. Alle Daten zwischen den Services müssen über verschlüsselte Verbindungen ausgetauscht werden. HTTP Anfragen an den CloudService erfolgen ausschliesslich über HTTPS.

Alle Anfragen an die API des Cloudservice müssen zudem mit einem entsprechenden JWT Token authentifiziert sein. Praxismitarbeitende werden durch den Cloudservice mittels Basic Authentication authentifiziert. Bei der Anmeldung mit Benutzername und Passwort liefert der Cloudservice ein JWT Token welches für weitere Anfragen an den Cloudservice weiterverwendet wird. Sowohl die Credentials für die Basic Authentication als auch das JWT Token werden durch die iOS Applikation im Keystore des Betriebssystems gespeichert. Das JWT Token wird regelmässig erneuert, indem die Basic Authentication mit den gespeicherten Credentials wiederholt wird. Der Ablauf für Authentifizierung wird damit unverändert aus dem Vorgängerprojekt übernommen [2].

Langfristig sollen Authentifizierung und Authorisierung mit OAuth und OpenId-Connect umgesetzt werden. Diese Integration von OAuth und OpenId-Connect ist nicht Bestandteil dieser Projektarbeit.

7.3 Sprachsynthese

Dieses Kapitel beschreibt die Integration der Sprachsynthese in das Praxisrufsystem. 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. Insofern Sprachbenachrichtigungen in den Einstellungen aktiviert sind, werden Benachrichtigungen, welche dieses Flag aktiviert haben. Abbildung 7.9 zeigt einen Ausschnitt aus dem Entity Relationship Diagramm der Domäne Configuration. Dabei sind die Felder, welche für die Sprachsynthese ergänzt werden, grün markiert.

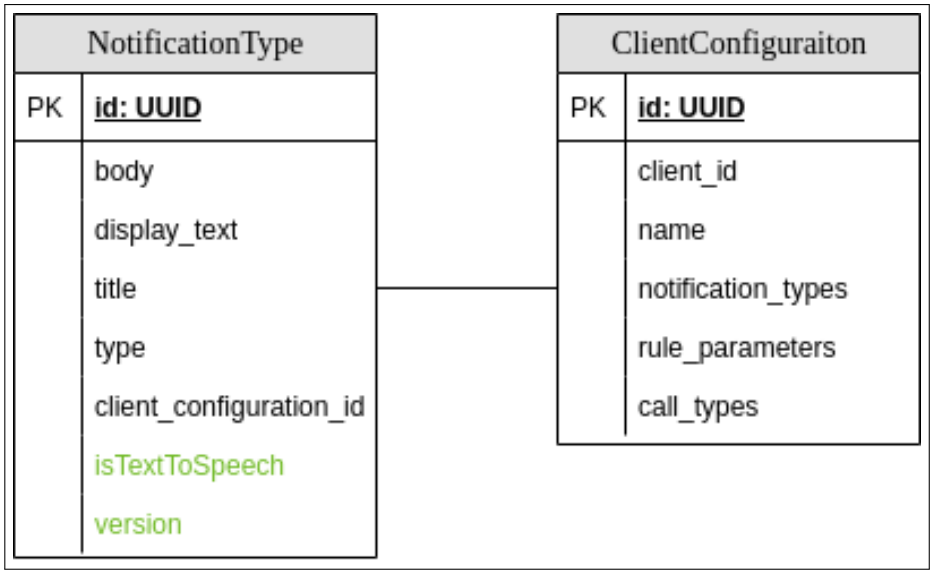


Abbildung 7.9: ERD Ausschnitt - Konfiguration Sprachsynthese

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 Integration Sprachsynthese Provider

Dieses Kapitel beschreibt, wie Amazon Polly in das cloudbasierte Praxisrufsystem integriert 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 REST API des entsprechenden Domänen-Moduls gehen. Diese Unabhängigkeit ermöglicht es, das Modul in Zukunft einfach aus dem Cloudservice auszubauen und als eigenständigen Microservice zu betreiben.

Die Abhängigkeit zu Amazon Polly als Provider 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 der Komponente, welche die Schnittstelle nach aussen bietet verwendet. Um einen neuen Provider zu unterstützen, kann dieses Interface implementiert werden und die neue Implementation in Schnittstelle verwendet werden. Das Klassendiagramm in Abbildung 7.10 gibt einen Überblick über den Aufbau des Moduls Speech Synthesis.

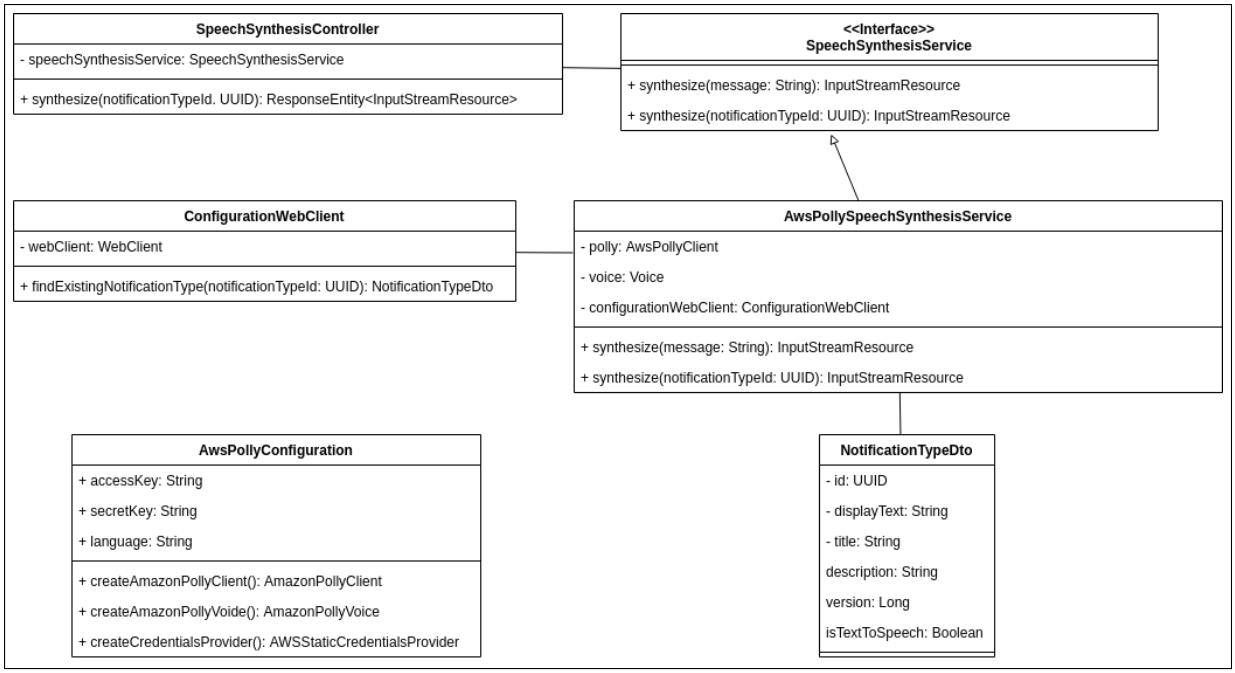


Abbildung 7.10: Klassendiagramm Modul SpeechSynthesis

Für die Anbindung des Providers Amazon Polly wird das SpeechSynthesisService Interface 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 instanziieren. Über die Dependency Injection von Spring werden diese Komponenten dem AwsPollySpeechSynthesisService übergeben werden.

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

7.3.3 Schnittstelle Cloud Service

Endgeräte im Praxisrufsystem 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 der Endpunkt die Abfrage Sprachdaten für die aktuellste Version eines Benachrichtigungstypen (NotificationType).

Für die Sprachsynthese wird das Feld ”title” auf aus der Entität NotificationType aus der Domäne Configuration verwendet. Der Endpunkt zum Bezug von Sprachdaten nimmt deshalb einen Parameter mit dem Namen ’notificationTypeId’ entgegen. Dieser Parameter muss dem technischen Identifikator eines Benachrichtigungstyps (NotificationType) entsprechen. Anhand dieses Identifikators wird der entsprechende NotificationType über die API des Moduls Configuration bezogen. Anschliessend wird eine Anfrage an Amazon Polly gesendet um die Textdaten als Sprache zu synthetisieren. 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 dabei als Binärdaten mit Media Type ”audio/mp3” im Body der Response zurückgegeben. Der Endpunkt wird aus Endgeräten mit Http GET Anfragen angesprochen.

7.3.4 Anbindung Mobile Client

Dieses Kapitel beschreibt wie die API des Cloudservice aus einer nativen iOS Applikation abgesprochen werden kann.

Für die Integration des SpeechSynthesis Endpunkts aus dem CloudService 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 unterstützten. Dazu wird die Komponente URLSession aus der iOS Standardbibliothek verwendet. Diese bietet mit URLSession.downloadTask die Möglichkeit Inhalte von einer URL herunterzuladen [32].

Der Service PraxisrufApi wird dementsprechend 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 Abfrage zu versenden. Die Resultate der Abfrage 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>” gespeichert. Sowohl NotificationTypeId als auch Version 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.

7.3.5 Laufzeitsicht

Dieses Kapitel fasst den Ablauf für das Senden, Empfangen und Vorlesen von Benachrichtigungen zusammen.

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 AppDelegate 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.

Nachdem eine empfangene Benachrichtigung der Inbox hinzugefügt wurde, wird geprüft ob Sprachsynthese in den lokalen Einstellungen aktiviert ist. Ist diese deaktiviert, endet die Verarbeitung. Ist die Sprachsynthese lokal aktiviert, 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 und Version der Benachrichtigung vorhanden ist. Ist dies der Fall, werden die Inhalte dieser Datei abgespielt und es wird keine Anfrage an den Cloudservice 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 die Inhalte der relevanten 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.

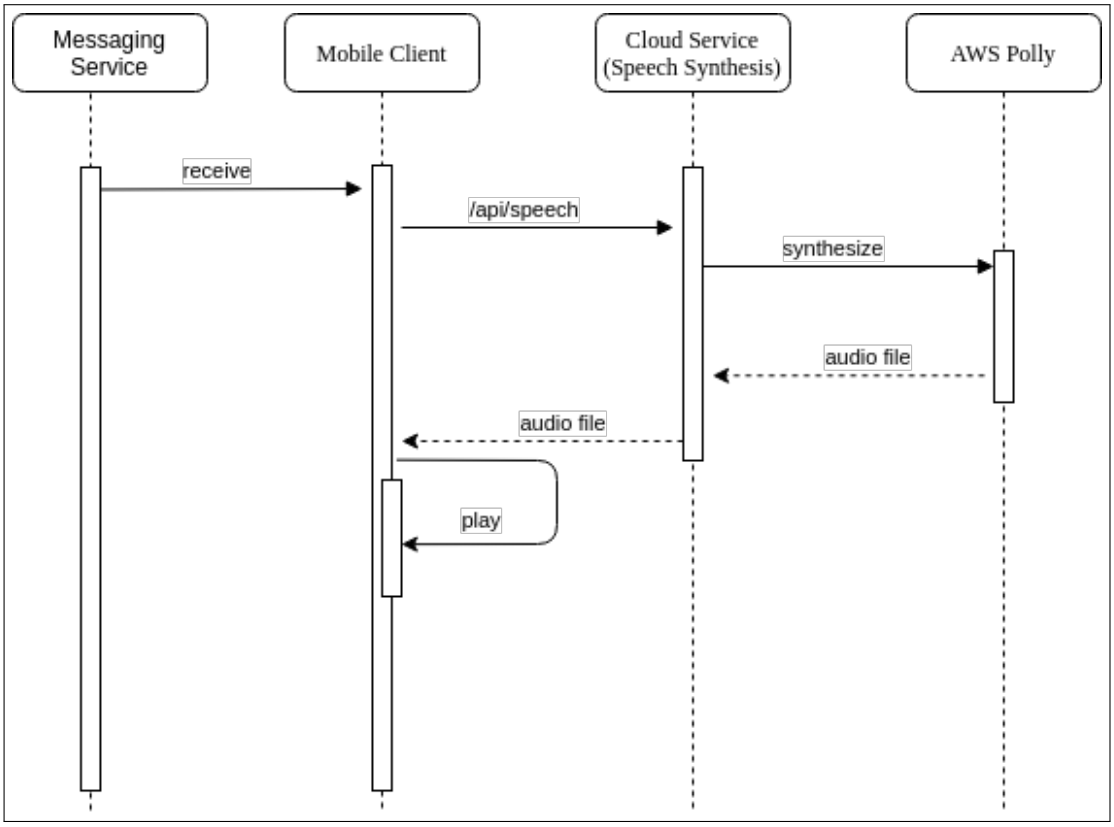


Abbildung 7.11: Sequenzdiagramm Sprachsynthese - Systemsicht

Abbildung 7.11 Stellt den Ablauf für das Empfangen und Vorlesen von Benachrichtigungen aus Systemsicht dar. Die Prüfung des lokalen Cache wird dabei nicht dargestellt.

7.4 Gegensprechanlage

Mit dem Einbau von synchroner Sprachübertragung wird das Praxisrufsystem um die Funktion Gegensprechanlage erweitert. Die gewählte Technologie WebRTC erlaubt es Peer-to-Peer Sprachverbindungen zwischen Clients aufzubauen. Dieses Kapitel beschreibt wie das Praxisrufsystem erweitert wird, um eine konfigurierbare Gegensprechanlage mit WebRTC zu integrieren.

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 der Domäne Configuration 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 7.12 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.

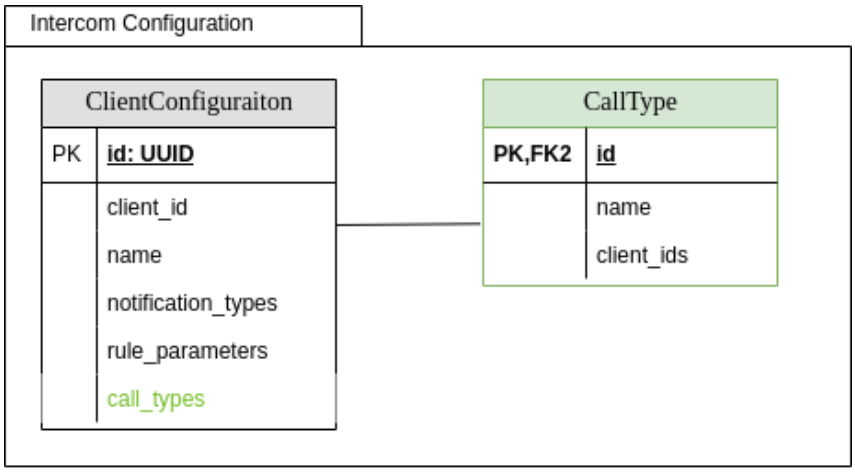


Abbildung 7.12: ERD Ausschnitt - Konfiguration Gegensprechanlage

Das Admin UI wird mit Ansichten erweitert, um CallTypes zu erstellen, anzeigen, bearbeiten und löschen. 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 bestehenden Endpunkte für ClientConfiguration werden entsprechend erweitert.

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.

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.

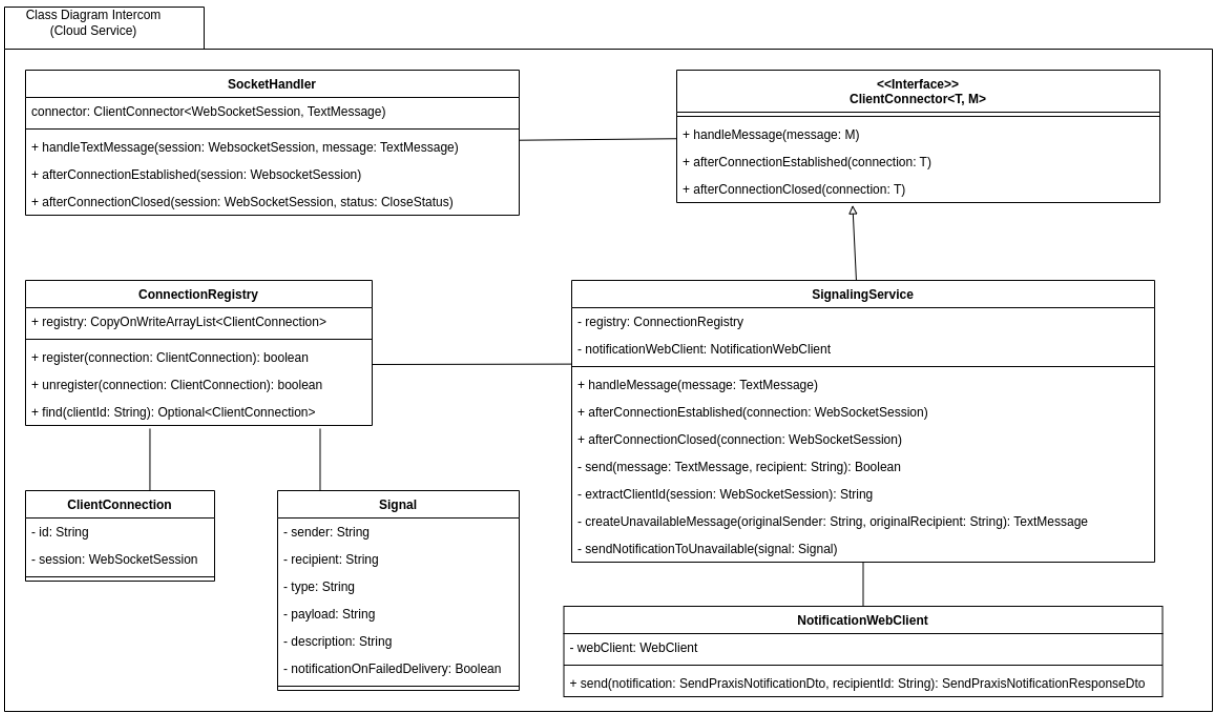


Abbildung 7.13: Klassendiagramm SpeechSynthesisController

Die Klasse SignalingService implementiert das ClientConnector Interface. Der SignalingService führt eine Liste an verfügbaren Verbindungen. Dabei wird jede Verbindung durch eine eindeutige Id gekennzeichnet. Für die Verwaltung von Verbindungen wird die Komponente ConnectionRegistry implementiert. Diese muss Verbindungen anhand einer Id registrieren und wieder entfernen können. Weiter müssen Verbindungen anhand ihrer Id gefunden werden können. Für das Zustellen von Signalen über bekannte Verbindungen wird die Methode handleSignal implementiert. Jedes Signal muss die Identifikation seines Empfängers beinhalten. 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 Schnittestelle 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 aufgerufen werden. Diese delegieren das Verwalten der Verbindungen an den SignalingService. Empfangene Meldungen werden an den SignalingService übergeben und wie oben beschrieben verarbeitet.

7.4.3 Signaling Security

Der Zugriff auf den Signaling Service 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 in den anderen Cloudservice Domänen verwendet werden. Über die SecurityConfig 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. Wenn kein gültiges Token vorhanden ist, wird eine entsprechende Fehlermeldung zurückgegeben. Der Aufbau der Websocketverbindung wird abgebrochen.

Mit der Klasse HttpSessionHandshakeInterceptor kann vor dem Aufbau einer Websocketverbindung geprüft werden, dass der Http Request für den Verbindungsaufbau authentifiziert wurde. Mit der Verarbeitung des Http Requests durch die SecurityConfig des Cloudservices wurden die Rollen, welche dem Aufrufer zugeteilt ausgelesen. Im HttpSessionHandshakeInterceptor können diese Informationen ausgewertet und geprüft werden, dass ein Benutzer für den Austausch von Signalmeldungen authorisiert ist. Ist er dies nicht, wird der Aufbau der Websocketverbindung wird abgebrochen und eine Fehlermeldung zurückgegeben. Das Praxisrufsystem kennt die zwei Rollen "ADMIN" und "USER". Beide Rollen sind berechtigt, dass Rufsystem über den Mobile Client zu verwenden und dürfen damit Signalmeldungen austauschen.

Für den Aufbau von Websockets wird das ausschliesslich das Protokoll Secure WebSockets (WSS) verwendet. Der Austausch von Signalmeldungen ist damit verschlüsselt.

7.4.4 Anmeldung und Registrierung

Der Ablauf von Anmeldung und Registrierung funktioniert mit dem neuen nativen Mobile Client grundsätzlich gleich wie zuvor. 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 Antowrt auf die Anmeldung gibt der Cloudservice ein JWT Token zurück. Dieses wird für die Authentifizierung aller 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.

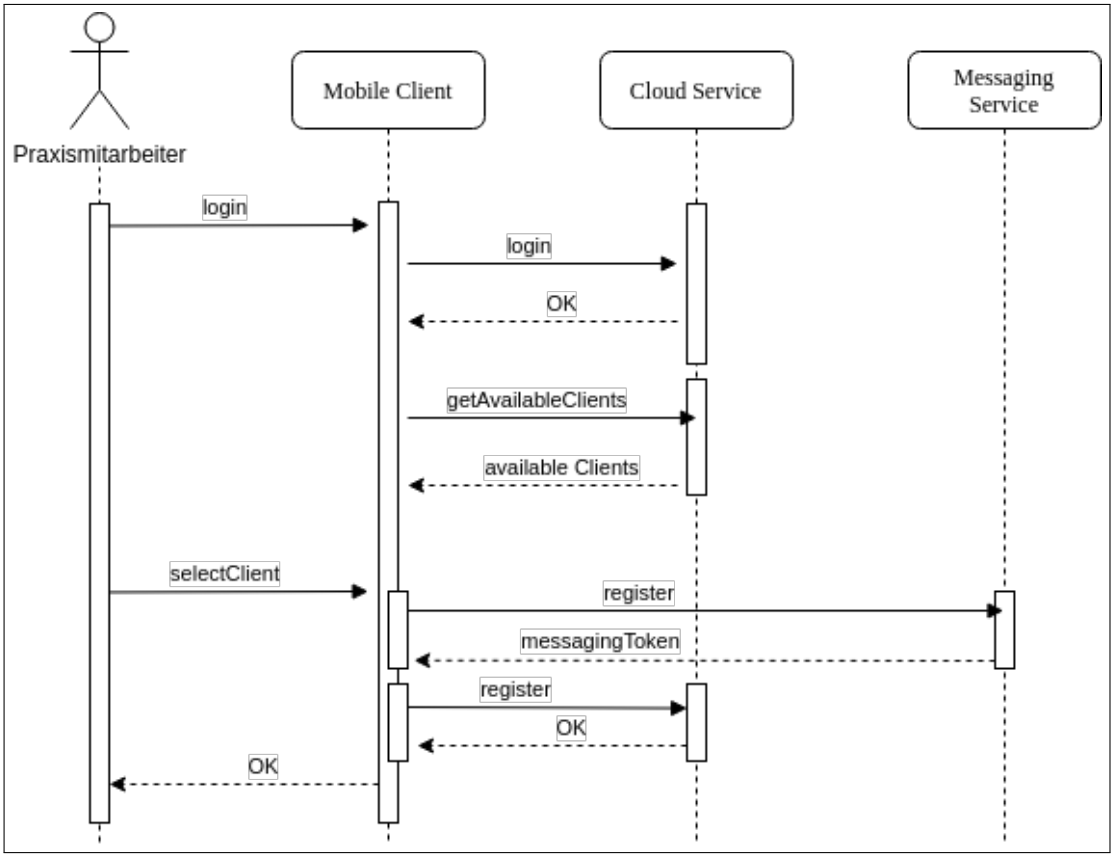


Abbildung 7.14: Sequenzdiagramm - Anmeldung und Registrierung im Mobile Client

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 für Sprachverbindungen verwendet werden.

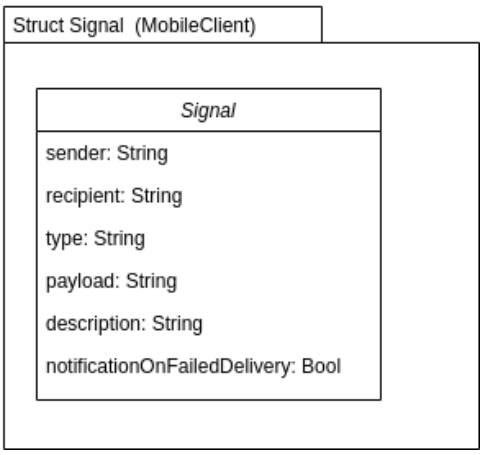


Abbildung 7.15: Signal (Mobile Client)

Alle Signalmeldungen beinhalten Identifikation von Sender und Empfänger. Diese werden vom Signalingserver verwendet, um die Signale korrekt weiterzuleiten. 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 TRUE ist, wird der Empfänger mit einer asynchrone Benachrichtigung darüber informiert. Dazu wird das bestehende Notification Modul des Cloudservices verwendet. Für die Verwaltung von Sprachverbindungen werden insgesamt sechs Typen von Signalmeldungen definiert.

7.4.6 Verbindungsaufbau

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, welche Clients mit dem betätigten Button angesprochen werden sollen. Der Cloudservice bietet dazu einen Endpoint an über den der vollständige CallType des verwendeten Buttons hinterlegt sind geladen werden können. Nachdem diese Informationen geladen sind, können Sprachverbindungen zu allen relevanten Clients 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 relevanten Empfänger und leitet die Signale über die jeweilige Verbindung weiter. Die Verbindung auf Empfängerseite wird nach Eingang des Offers initialisiert. Danach wird eine Answer zurück an den Initiator gesendet, welche über den Signalingserver zugestellt wird, Der Initiator empfängt die Antwort Signale und ergänzt die notwendigen Verbindungsinformationen. Folgende Graphik visualisiert diesen Ablauf: **TODO:** Add ICE Candidate Exchange

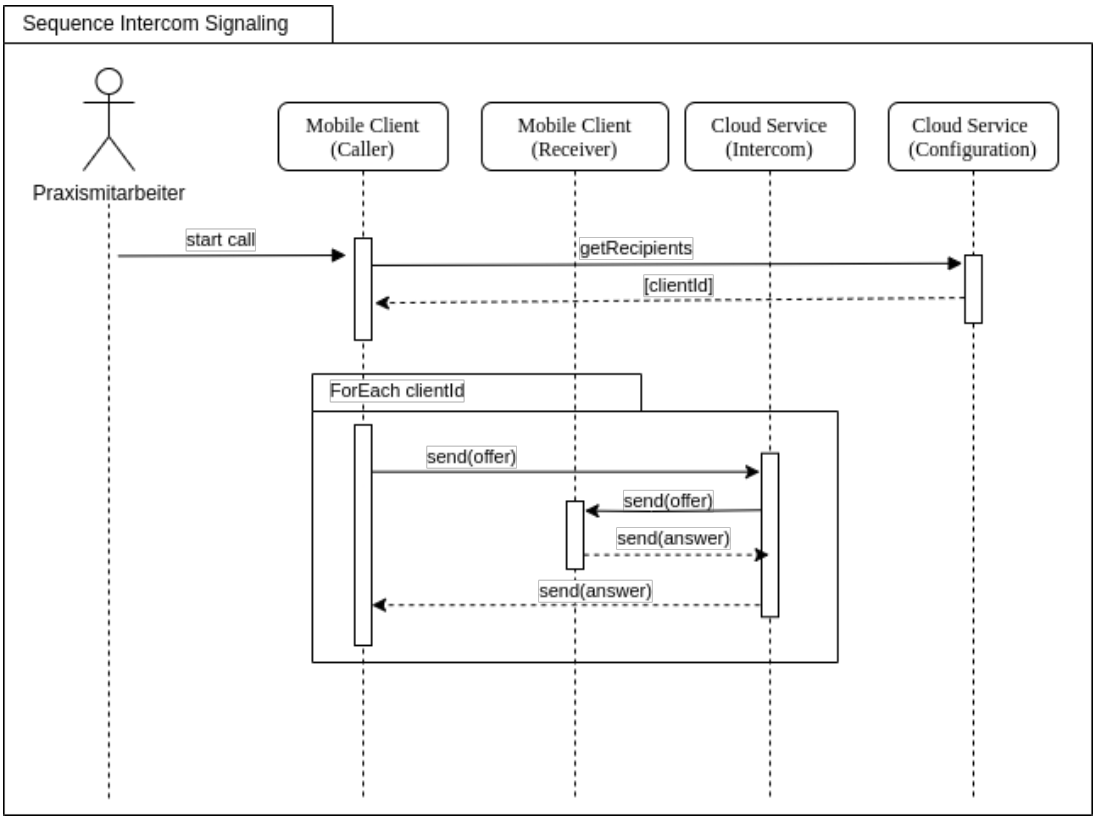


Abbildung 7.16: Ablauf Verbindungsaufbau Gegensprechanlage

Nachdem die Offer und Answer Meldungen ausgetauscht sind, müssen Ice Candidate Meldungen versendet werden. So können sich die CLients über Routing- und Protokollinformationen einigen die für die Peer to Peer Verbindung verwendet werden. Der Austausch von Ice Meldungen wird in obiger Graphik nicht angezeigt. Er verläuft nach demselben Prinzip wie der Austausch von Offer und Answer. Sobald sich die Clients auf die Verbindungsinformationen geeinigt haben, besteht die Sprachverbindung.

7.4.7 Anbindung Mobile Client an Singaling Server

Für den Aufbau von Sprachverbindungen zwischen Mobile Clients müssen mehrere Signalmeldungen ausgetauscht werden. Der Cloud Service wird im Rahmen dieses Projektes um eine Websocket Schnittstelle erweitert, welche dies ermöglicht.[3] Als Technologie für diese Schnittstelle werden Websockets verwendet. Der Api Service wird dementsprechend erweitert, um Websocket Verbindungen zu ermöglichen. Dies beinhaltet den Auf- und Abbau von Websocket Verbindungen, sowie das Senden und Empfangen von Meldungen über diese Verbindung. Weiter muss die Verbindung konstant offen gehalten und im Fehlerfall erneut aufgebaut werden können.

Der Austausch von Signalmeldungen ist der einzige Anwendungsfall in Praxisruf, der Websocketverbindungen benötigt. Deshalb wird auf eine generische Integration von Verbindungen analog von Http Verbindungen[4] verzichtet. Stattdessen wird eine Extension Klasse PraxisrufApi+Signaling nach folgendem Muster erstellt:

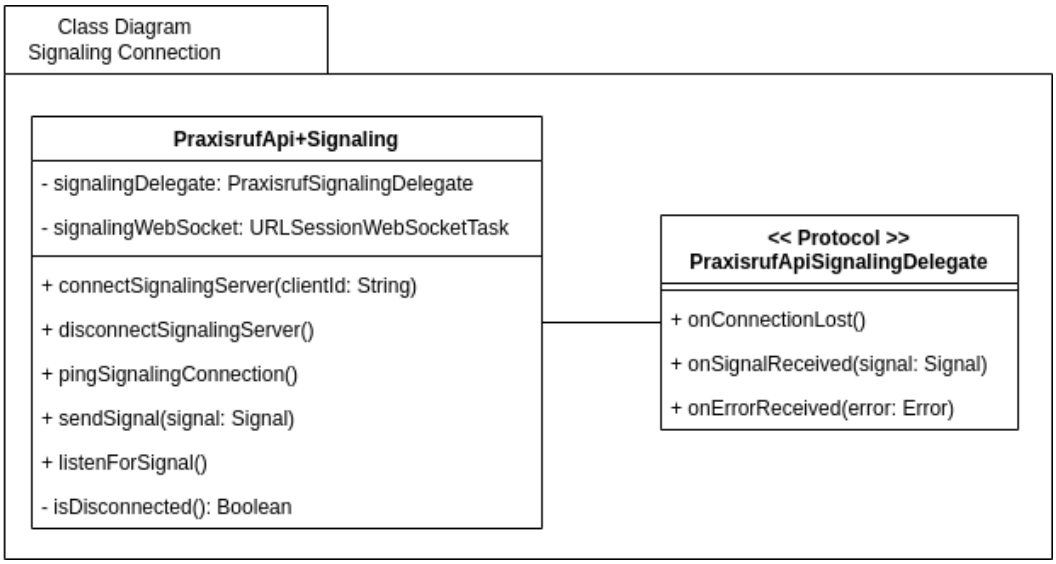


Abbildung 7.17: Klassendiagramm - Signaling Schnittstelle in Mobile Client

Die Signaling Extension kann immer genau eine offene Verbindung verwalten. Diese wird als Instanzvariable innerhalb des Services geführt. Der Status dieser Verbindung kann als Computed Property ”disconnectedïntern abgefragt werden. Die Extension bietet Methoden um den Signaling Service zu Verbinden. Eine Verbindung wird dabei immer spezifisch für die aktuell gewählte Client Configuration geöffnet. [5] Weiter werden Methoden angeboten, um Pingsignale zu senden oder die Verbindung zu trennen. Zudem bietet die Extension Methoden, um Signalmeldungen über die geöffnete Verbindung zu senden, sowie eine Methode um empfangene Signalmeldungen zu verarbeiten. Für die Integration der Singalingverbindung in den Rest der Applikation, wird das Protokoll PraxisrufApiSignalingDelegate definiert.

Vor dem Versenden einer Signal- oder Pingmeldung wird überprüft, ob eine Verbindung geöffnet und Fehlerfrei ist. Ist dies nicht der Fall, wird die Verarbeitung die onConnectionLostMethode des Delegates aufgerufen. Die Implementation dieses Delegates ist dafür verantwortlich, die Verbindung wieder zu öffnen. Die Methoden onSignalReceived resp. onErrorReceived werden aufgerufen wenn ein Signal resp. eine Fehlermeldung empfangen wurde. Im Fehlerfall soll wenn nötig eine Fehlermeldung angezeigt werden und die Verbindung repariert werden. Wenn eine Signalmeldung empfangen wurde, muss diese an die Applikation zu Verarbeitung weitergereicht werden.

[3] Vgl. Kapitel 5.4
[4] Vgl. Kapitel 5.2.2
[5] Vgl. Kapitel X.X

7.4.8 Signalverarbeitung im Mobile Client

Neben austausch von Signalen muss auch die Sprachverbindung selbst verwaltet werden. Die Details der Sprachverbindung sind vendor spezifisch. Es wird deshalb eine eigene Klasse CallClient erstellt. Diese ist für das Verwalten von Verbindungen verantwortlich. Bei eingehenden Verbindungen muss sie signale empfangen, die Verbindung erstellen. Wenn nötig Antowrt Signale erstellen und zurückgeben. Bei ausgehenden Verbindungen muss sie Verbindung initialisieren. Es muss Signal erstellt und versendet werden. Weiter müssen Methoden angeboten werden um die Unterhaltung oder das Microphon zu muten. Und um die Verbindung zu schliessen. Und um den bei Änderung des internen Status, das UI entsprechend zu aktualisieren.

Um die Integration der Sprachverbindung möglichst unabhängig und auswechselbar zu machen, wird der CalLClient nicht direkt in der View verwendet. Stattdessen definiert der CallClient ein Delegate Protocol, welches die notwendigen Callbacks definiert.

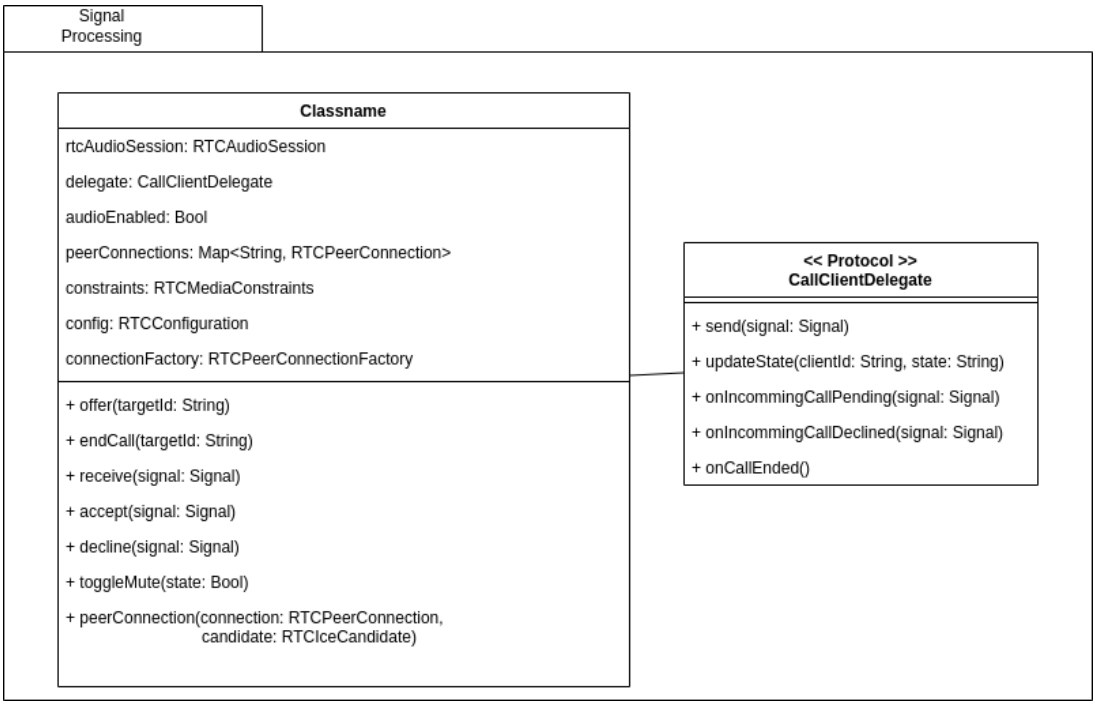


Abbildung 7.18: Klassendiagramm - Signalingverarbeitung in Mobile Client

Um Anrufe in der Applikation verwenden zu können müssen CallClient und PraxisrufApi+Signaling in die Benutzeroberfläche integriert werden. Beide Applikationen definieren ein Delegate Protocol, welches die Funktionen spezifiziert, über welche die Komponenten eingebunden werden können. Es wird eine weitere Serviceklasse mit dem Namen CallService implementiert, welche beide Delegate Protocols implementiert. Dieser Service instanziert CallClient und PraxisrufApi+Signaling und registriert sich anschliessend als Delegate bei beiden Instanzen. Der CallService selbst wird in der View verwendet. Er nimmt Benutzereingaben entgegen und delegiert die entsprechende Funktionalität an den CallClient und SignalingClient.

Wenn der Benutzer 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 Signel, um den Empfänger zu informieren. Dieses Signal gibt er an den CallService weiter. Der CallService leitet das Signal an den SignalingClient weiter, welcher das Versenden an den Cloud Service übernimmt.

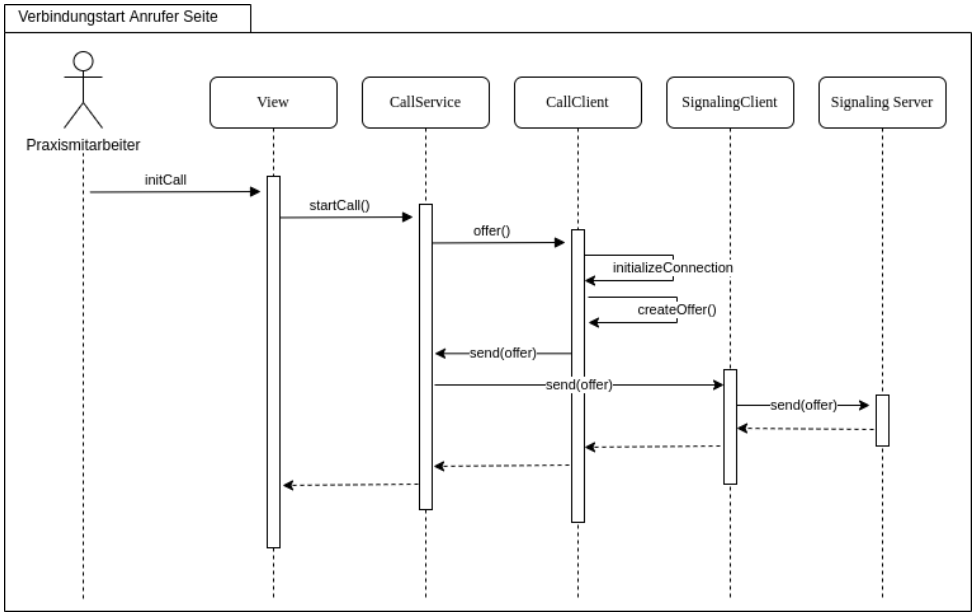


Abbildung 7.19: MobileClient - Anruf Starten Signal

Das versendete Signal wird über das Signaling Modul des Cloud Service an den Empfänger übermittelt. Dieser empfängt das Signal über den SingalingClient. Der SignalingClient gibt das Signal über onSignal-Received an den CallService weiter. Der CallService aktiviert die Ansicht für aktive Anrufe und leitet das Signal an den CallClient weiter. Der CallClient initialisiert die lokalen Verbindungsinformationen und erstellt eine Signal zur Bestätigung. Dieses Signal wird wiederun über den CallService zum Signaling-Client weiter zum Cloud Service versendet. Auf Starterseite, kann diese Bestätigung weiterverarbeitet werden.

7.4.9 Verpasste Anrufe

Anrufe über die Gegensprechanlage können nur empfangen werden, solange die Praxisruf App im Vordergrund läuft und beim Signalingserver registriert ist. Ist dies nicht der Fall, kann der Signalingserver keine Signale an den jeweiligen Empfänger zustellen. Wenn der Singalingserver ein Signal für einen Empfänger erhält, der nicht verbunden ist wird versucht, diese über eine Benachrichtigung zu informieren. Benachrichtigungen für verpasste Anrufe, werden gleich wie alle anderen Benachrichtigungen empfangen und in der Inbox angezeigt. Für das Versenden der Benachrichtigung für verpasste Anrufe wird das bestehende Notification Modul des Cloudservice verwendet. Dieses wird um einen Endpunkt erweitert, der es erlaubt Benachrichtigungen gezielt an einen einzelnen Client zu versenden. Der neue Enpunkt nimmt zwei Parameter entgegen: Die technische Identifikation des Empfängers und die technische Identifikation des relevanten Benachrichtigungstyps. Der relevante Benachrichtigungstyp muss durch den Praxisadministrator im Admin UI erfasst werden. Die technische Identifikation dieses Benachrichtigungstyps muss anschliessend in den Umgebungsvariablen des Signalingservices hinterlegt werden.[6] Wenn der Empfänger nicht für Benachrichtigungen registriert ist, kann er nicht über den verpassten Anruf informiert werden. Dasselbe gilt, wenn die Benachrichtigung aus technischen Gründen nicht zugestellt werden kann. Im Rahmen dieses Projektes wird kein Mechanismus implementiert, um diese fehlgeschlagene Zustellung automatisch zu wiederholen.

7.4.10 Deaktivierte Anrufe

Praxismitarbeitende können das Empfangen von Anrufen lokal deaktivieren. Wenn der Empfang von Anrufen deaktiviert ist, werden Offer Signale nicht mit einer Answer sondern mit einem Declined Signal beantwortet. Empfängt ein Client ein Declined Signal, erstellt er lokal einen Eintrag in der Inbox und zeigt eine Push Benachrichtigung an um den Benutzer darauf hinzuweisen.

7.4.11 Verbindungsende

Praxismitarbeitende können Sprachverbindungen durch einen Button beenden. Durch antippen des Auflagen-Buttons wird die bestehende Peer to Peer Verbindung zum Gesprächspartner getrennt. Anschliessend wird ein Signal vom Typ End an die Gesprächspartner gesendet. Durch Empfang des End-Signals, weiss der Gesprächspartner, dass die Verbindung durch das Gegenüber getrennt wurde und kann die Verbindung seinerseits entfernen.

[6] Vgl. Installationsanleitung

7.5 Zusammenfassung

7.5.1 Mobile Client

TODO: Add App Delegate **TODO:** Add Views

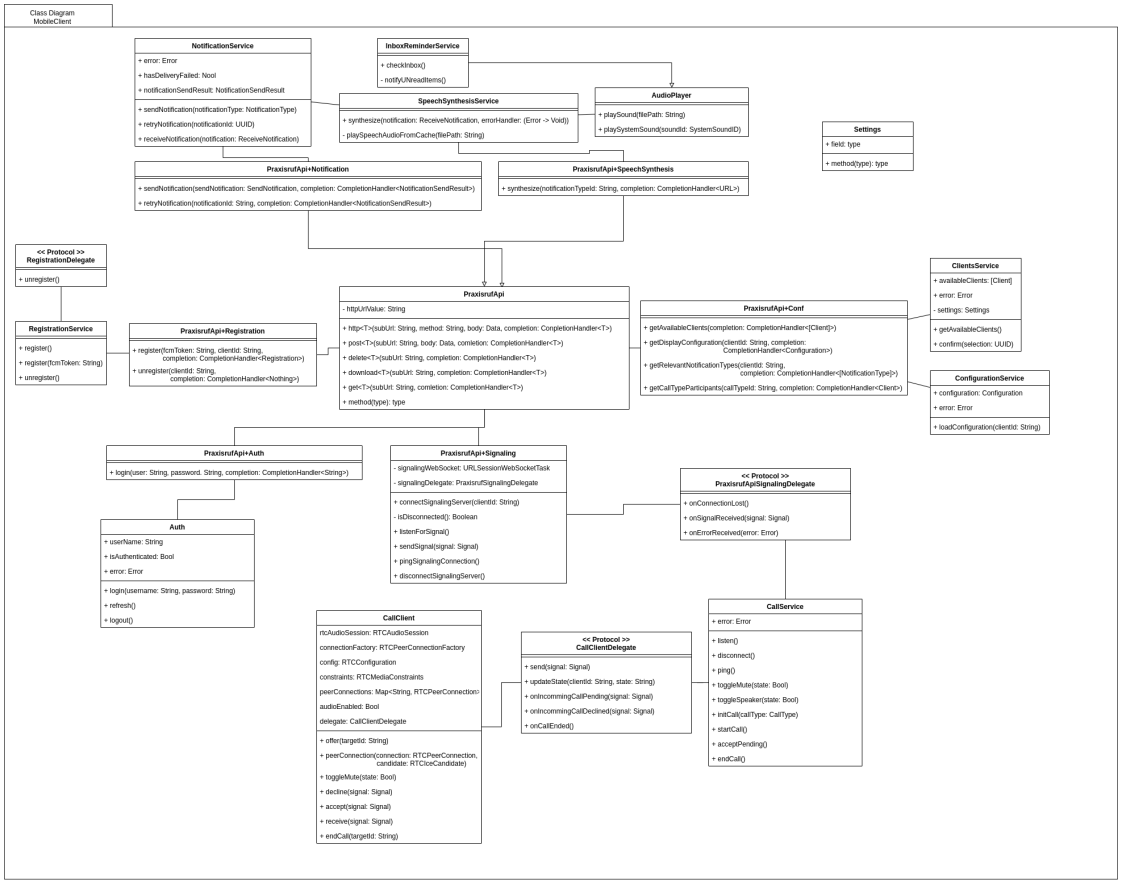


Abbildung 7.20: Klassendiagramm Modul SpeechSynthesis

7.5.2 Cloudservice Entity Relation Diagramm

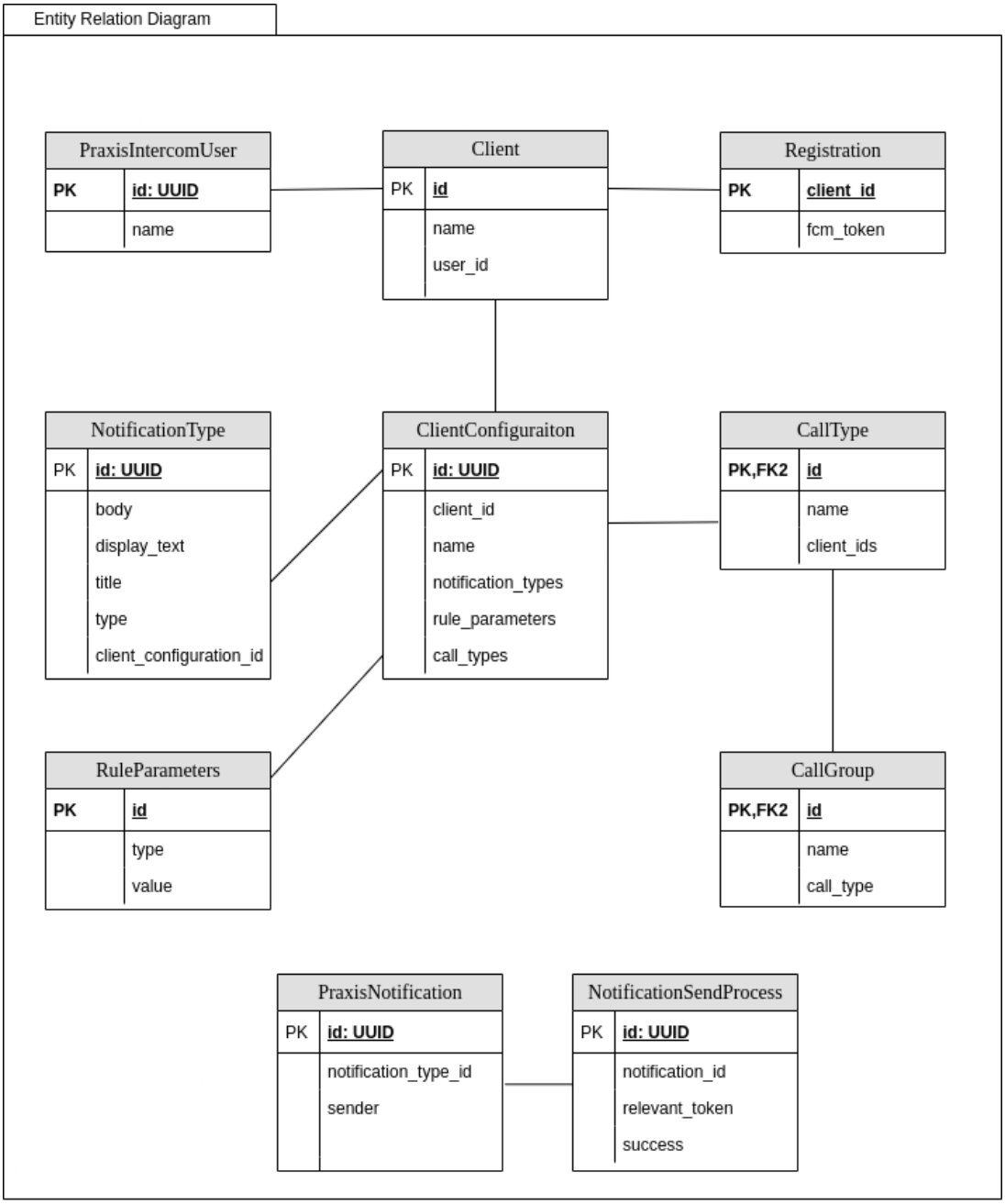


Abbildung 7.21: Entitiy Relation Diagramm - Cloudservice

7.5.3 Cloudservice Modul Speech Synthesis

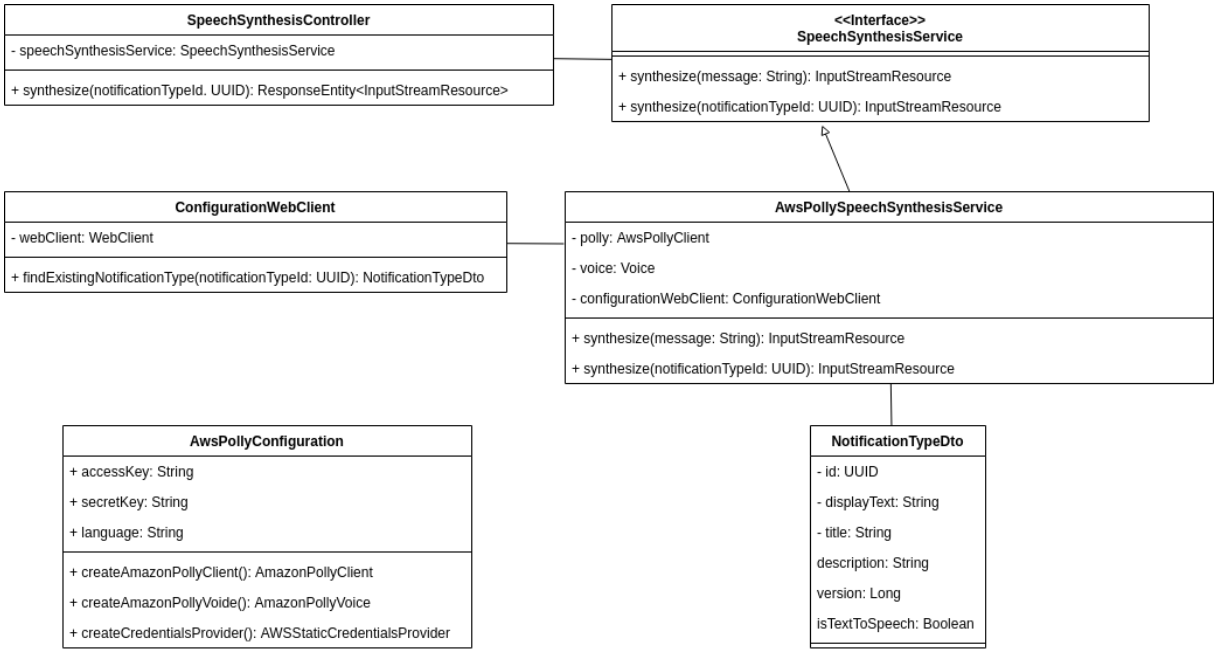


Abbildung 7.22: Klassendiagramm Modul SpeechSynthesis

7.5.4 Cloudservice Modul Intercom

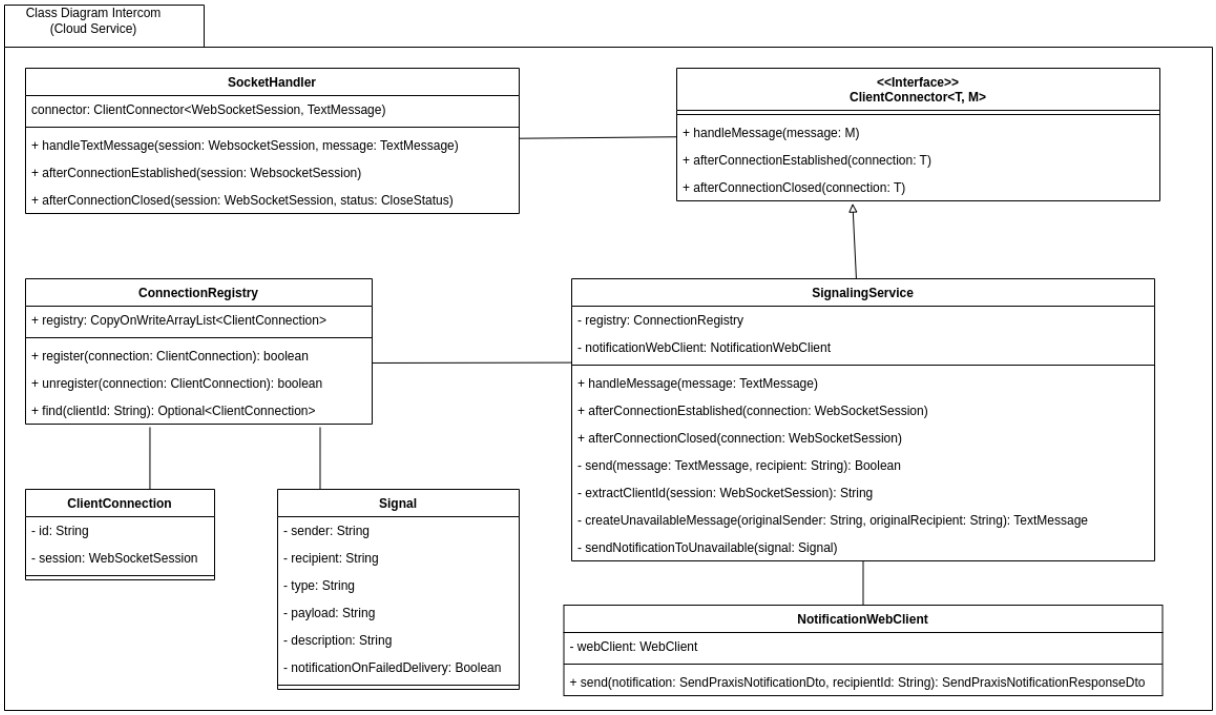


Abbildung 7.23: Klassendiagramm Modul Intercom

7.5.5 Cloudservice Schnittstellen

Der Cloudservice wird um folgende Endpoints erweitert:

Aktion	HTTP	Pfad	Body	Response
Alle CallTypes laden	GET	/api/config/calltype	-	[CallTypeDto]
CalLType laden	GET	/api/config/calltype/id	-	CallTypeDto
CallType erstellen	POST	/api/config/calltypes	CallTypeDto	CallTypeDto
CallType aktualisieren	PUT	/api/config/calltypes	CallTypeDto	CallTypeDto
CallType löschen	DELETE	/api/config/calltypes/id	-	-
Mehrere CallTypes löschen	DELETE	/api/config/calltypes/many/filter	-	-
Alle CallGroups laden	GET	/api/config/callgroup	-	[CallGroupDto]
CallGroup laden	GET	/api/config/callgroup/id	-	CallGroupDto
CallGroup suchen	GET	/api/config/callgroup?callTypeId	-	[CallGroupDto]
CallGroup erstellen	POST	/api/config/callgroup	CallGroupDto	CallGroupDto
CallGroup aktualisieren	PUT	/api/config/callgroup	CallGroupDto	CallGroupDto
CallGroup löschen	DELETE	/api/config/callgroup/id	-	-
Mehrere CallGroups löschen	DELETE	/api/config/callgroup/many/filter	-	-
Sprachsynthese für notificationType	GET	/api/speech/id	-	MP3 Datei

Zudem werden die bestehenden Endpoints zur Verwaltung von NotificationType und ClientConfiguration Daten erweitert. Sodass neu CallTypes auf ClientConfigurations registriert werden können und das isTextToSpeech Flag auf NotificationTypes gesetzt werden.

Letztlich wird ein neuer Websocket Endpoint unter /api/intercom/signaling veröffentlicht.

8 Umsetzung

8.1 Resultate

Praxisruf wurde wie in Kapitel 5 beschrieben erweitert. Es wurde eine native iOS App entwickelt, welche den Funktionsumgang des bestehenden Mobile Clients vollständig unterstützt. Weiter wurde mit AWS Polly ein Sprachsynthese Provider an das System angebunden. Diese Anbindung wird verwendet, um bei Bedarf den Inhalt empfangener Benachrichtigungen automatisch vorlesen zu lassen. Letztlich wurde WebRTC verwendet, um eine konfigurierbare Gegensprechanlage zu implementieren. Sowohl das Vorlesen von Benachrichtigungen, als auch die Gegensprechanlage sind über das Admin UI konfigurierbar.

8.1.1 Systemarchitektur

Die Systemarchitektur von Praxisruf wurde im Rahmen dieses Projektes erweitert. Betroffen davon sind der Mobile Client und Cloudservice. Der Mobile Client wurde durch einen neuen nativen Mobile Client ersetzt. Der Cloudservice wurde um Komponenten zur Abfrage von Sprachdaten und Signalaustausch erweitert. Im Rahmen dieses Projektes wurde zudem der interne Aufbau des Cloudservices überarbeitet. Der Cloudservice wird nach wie vor als einzelner Service betrieben. Dieser Service ist aber neu in mehrere domänenspezifische Module aufgeteilt.[7] Diese Module können in Zukunft mit wenig Aufwand aus dem monolithischen Service ausgelöst und als eigene Services betrieben werden.

[7] Siehe Kapitel 5.1

8.1.2 Nativer Mobile Client

Dieses Kapitel zeigt die umgesetzten Ansichten des Mobile Clients.

Anmeldung und Konfiguration

Der Mobile Client bietet eine einfaches Verfahren zur Anmeldung und Konfiguration. In einem ersten Schritt gibt der Praxismitarbeitende Benutzername und Passwort ein. Anschliessend kann er auf einer zweiten Ansicht, die gewünschte Zimmerkonfiguration wählen. Benutzer und Zimmerkonfiguration werden dabei gespeichert. Bis sich der Benutzer manuell abmeldet wird bei allen zukünftigen Starts der App die gespeicherte Kombination von Benutzer und Zimmerkonfiguration wiederverwendet.

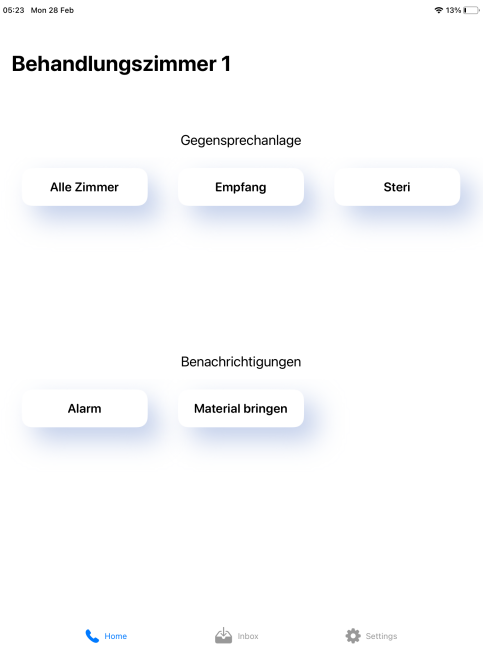


Abbildung 8.1: Ansicht Login

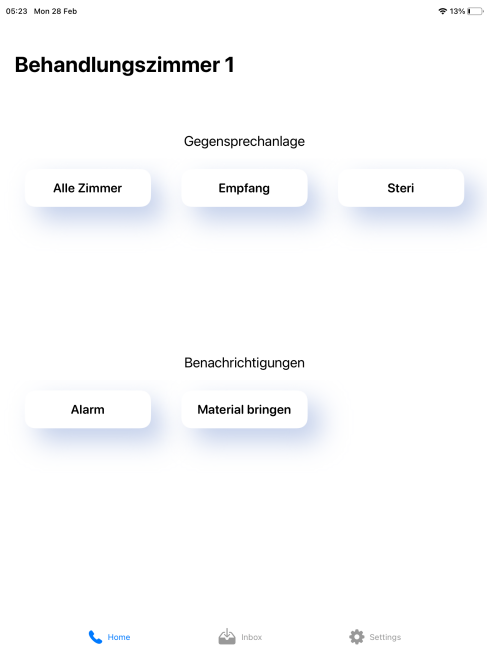


Abbildung 8.2: Ansicht Zimmerkonfiguration

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 Der Bereich Home ist in zwei Teile gegliedert und beinhaltet 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 vom Praxisadministrator konfiguriert. Der Bereich Inbox zeigt eine Liste von empfangenen Benachrichtigungen sowie verpassten und vergangenen Anrufen. Einträge in dieser Liste können durch eine Wischgeste (Swipe right) entfernt werden.

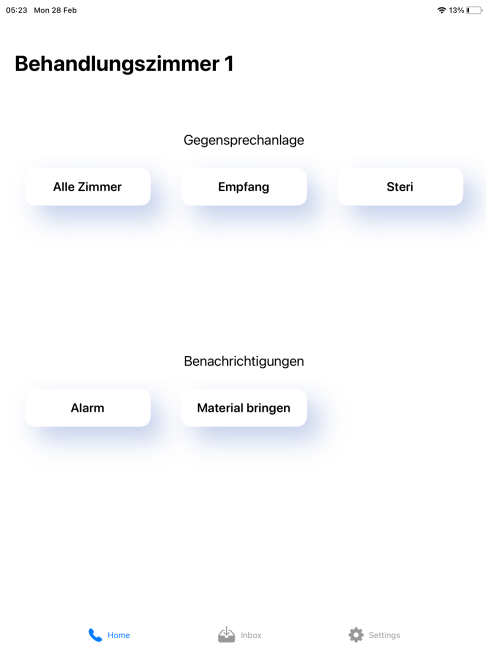


Abbildung 8.3: Ansicht Home

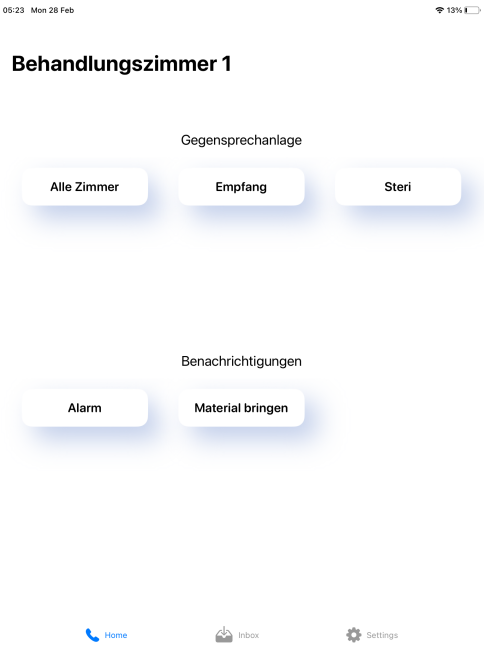


Abbildung 8.4: Ansicht Inbox

Einstellungen und Aktive Anrufe

Im Bereich Einstellungen 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. Über einen Button kann der Benutzer sich zudem von der App abmelden.

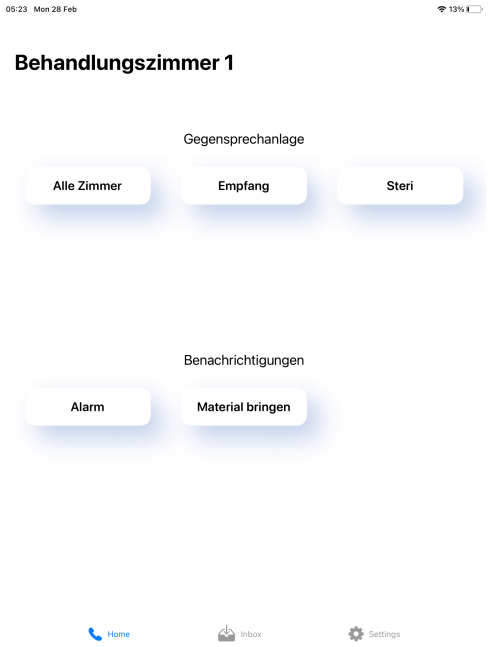


Abbildung 8.5: Ansicht Settings

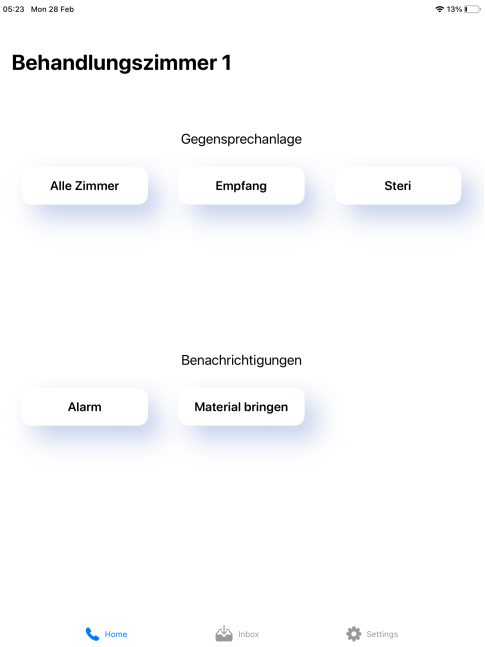


Abbildung 8.6: Ansicht Aktiver Anruf

Die Ansicht ”Aktive Anruf” wird angezeigt, nachdem ein Anruf gestartet wurde. Entweder, der Anruf durch antippen des Buttons in der Home Ansicht gestartet wurde oder weil ein Anruf von einem anderen Client empfangen wird. 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. Allen Gesprächsteilnehmern stehen Buttons zur Stummschaltung des eigenen Lautsprechers und Microphons zur Verfügung. Zudem können alle Gesprächsteilehmmer die Unterhaltung durch den roten Auflegen Button beenden.

Hintergrundbenachrichtigungen und Fehlerhandling

Das Versenden von Benachrichtigungen ist über die Anbindung des Cloudservices gelöst. Dieser leitet Benachrichtigungen anhand der Konfiguration über den Messagingservice an die relevanten Empfänger zu. Bei einem Fehler in der Verarbeitung beim Cloudservice, wird Praxismitarbeitenden direkt mitgeteilt, dass die Benachrichtigung nicht zugestellt werden konnte. In der App wird ein Dialog angezeigt, welcher darüber informiert und die Möglichkeit bietet, die Aktion direkt zu wiederholen.

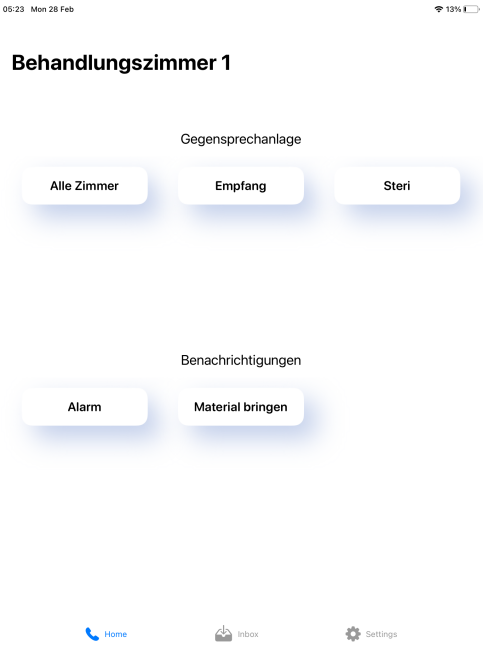


Abbildung 8.7: Hintergrund Benachrichtigung

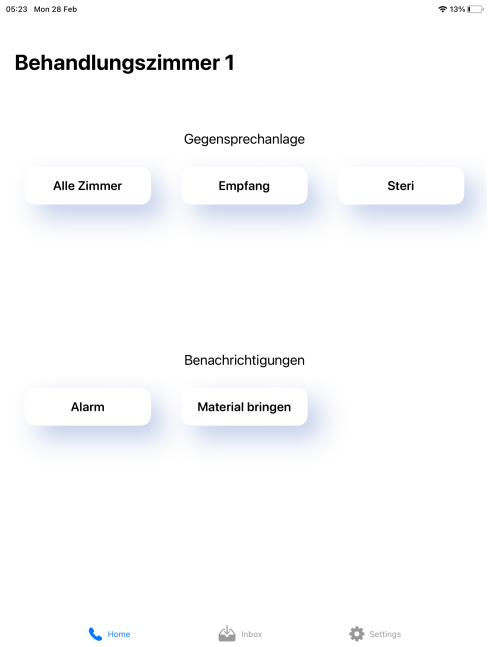


Abbildung 8.8: Benachrichtigung wiederholen

Benachrichtigungen können auch im Hintergrund empfangen werden. Im Hintergrund empfangene Benachrichtigungen erscheinen als Push Benachrichtigungen auf dem Home Screen des iPads. Anrufe über die Gegensprechanlage 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.

8.1.3 Sprachsynthese

Die Sprachsynthese für Benachrichtigungen in einem Praxisrufsystem konnten vollständig umgesetzt werden. Praxisadministratoren können über das Admin UI konfigurieren, welche Benachrichtigungen vorgelesen werden. Beim Empfang einer solchen Benachrichtigung im Mobile Client wird deren Inhalt automatisch vorgelesen. Die entsprechenden Sprachdaten bezieht der Mobile Client entweder vom Cloudservice oder aus einem lokalen Cache. Für die Konvertierung von Benachrichtigungen zu Sprachdaten wurde im Cloudservice eine Anbindung an den Sprachsyntheseprovicer AWS Polly implementiert. Der Cloudservice bietet nach aussen eine Http Schnittstelle über welche Sprachdaten von Benachrichtigungen bezogen werden können. Sprachdaten müssen immer anhand der technischen Identifikation des relevanten Benachrichtigungstypes angefragt werden. Die konkreten Daten für die Synthese werden anhand dieser Identifikation aus der persistierten Konfiguration geladen.

8.1.4 Gegensprechanlage

Es wurde eine konfigurierbare Gegensprechanlage entwickelt und in Praxisruf integriert. Die Konfiguration der Gegensprechanlage wurde in die bestehende Konfiguration eingebunden. Praxisadministratoren können über das Admin UI konfigurieren, welche Sprachverbindungen in welchen Zimmern zu Verfügung stehen. Diese Konfiguration wird bei der Anmeldung im Mobile Client geladen. Anhand der Konfiguration werden Buttons für Sprachverbindungen angezeigt.

Praxismitarbeitende können Sprachverbindungen zu anderen Clients über Buttons im Mobile Client starten. Während laufenden Sprachverbindungen sehen Praxismitarbeitende im Mobile Client den Namen aller Gesprächspartner und deren Verbindungsstatus. Gesprächsteilnehmende können ihre Sprachverbindungen stummschalten und beenden. Kann ein Gesprächspartner nicht erreicht werden, wird er mit einer asynchrone Benachrichtigunge über den verpassten Anruf informiert. Spracherbindungen auf Empfängerseite werden automatisch angenommen. Bevor die Sprachübertragung aktiviert ist, wird der Empfänger mit einem Benachrichtigungston auf die Verbindung aufmerksam gemacht. Praxismitarbeitende haben die dieses Verhalten in lokalen Einstellungen anzupassen und Sprachverbindungen zu deaktivieren. Durch diese Einstellungen, werden Sprachverbindungen immer abgelehnt. Es wird allerdings eine Benachrichtung angezeigt, welche über den verpassten Anruf informiert.

Sprachverbindungen werden für jeden Gesprächspartner als Peer To Peer Verbindung zwischen Anrufer und Empfänger aufgebaut. Um diesen Verbindungsaufbau zu ermöglichen, müssen Signale zwischen den Beteiligten ausgetauscht werden. Dazu wurde der Cloudservice um die Komponente Signaling erweitert. Diese ermöglicht es Signale über Websocketverbindungen zu versenden. Sie Signaling Komponenten verwaltet alle offenen Verbindungen und vermittelt Signale an die relevanten Empfänger. Kann ein Signal nicht zugestellt werden, wird der Empfänger über das Notification Modul mit einer Benachrichtigung informiert.

Die umgesetzte Gegensprechanlage erfüllt alle in Kapitel 3 erarbeiteten Anforderungen. Sie hat haber zwei Einschränkungen, welche hier erwähnt sein sollen: Die erste Einschränkung betrifft den Betrieb von Mobile Clients. Die Qualität der Sprachverbindungen ist ausreichend für eine Gegensprechanlage. Grundsätzlich ist es möglich, mehrere Endgeräte im selben Zimmer zu betreiben. Dabei ist aber darauf zu achten, dass diese nicht zu nahe beineinander stehen. Wird eine Sprachverbindung zwischen zwei Mobile Clients aufgebaut, die unmittelbar nebeneinander stehen kann ein positiver Feedback Loop entstehen, welcher zu einem schrillen Pfeifton führt. Praxisruf ist für ein Mobile Client pro Zimmer konzipiert. Diese Einschränkung ist für den Praxisbetrieb minimal bis gar nicht relevant. Die zweite Einschränkung betrifft den austausch von Sprachdaten. Sprachverbindungen können zwischen zwei oder mehr Empfängern aufgebaut werden. Die implementierte Gegensprechanlage hat dabei aber die Einschränkung, dass Verbindungen nur zwischen Anrufer und Empfänger aufgebaut wird. Es werden keine Verbindungen zwischen den einzelnen Empfängern aufgebaut. Die implementierte Gegensprechanlage unterstützt damit 1:1 und 1:n Verbindungen aber keine n:n Verbindungen.

8.2 Tests

8.2.1 Benutzertests

Lorem ipsum

8.2.2 Testplan migrierte Funktionalität

Im Rahmen des Projektes IP5 Cloudbasiertes Praxisrufsystem wurde ein finaler Testplan definiert. Diese Tests wurden zum Abschluss des Projektes durchgeführt, um die Funktionalität des Systems abschliessend zu testen. Um zu gewährleisten, dass alle Funktionen aus dem alten (Shared Platform) Mobile CLient auch im neuen (Native) Mobile Client zur Verfügung stehen wurden diese Tests zum Abschluss dieses Projekts wiederholt. Die detaillierte Definition der Testszenarien sind im Anhang E des Projektberichts IP5 Cloudbasiertes Praxisrufsystem beschrieben.[2] Folgendes Protokoll gibt eine Übersicht über die Tests

Folgendes Protokoll zeigt den Stand der letzten Ausführung der Tests am xx.xx.2022:

Szenario	**Beschreibung**	**Resultat**
S01	Benachrichtigung versenden - Empfänger konfiguriert	tbd
S02	Benachrichtigung versenden - kein Empfänger	tbd
S03	Benachrichtigung empfangen.	tbd
S04	Fehler beim Versenden anzeigen.	tbd
S05	Wiederholen im Fehlerfall bestätigen.	tbd
S06	Wiederholen im Fehlerfall abbrechen.	tbd
S07	Audiosignal bei Benachrichtigung.	tbd
S08	Push Benachrichtigung im Hintergrund.	tbd
S09	Erinnerungston für nicht Quittierte Benachrichtigungen.	tbd
S10	Start Mobile Client - nicht angemeldet	tbd
S11	Start Mobile Client - angemeldet	tbd
S12	Anmelden mit korrekten Daten.	tbd
S13	Anmeldung mit ungültigen Daten.	tbd
S14	Konfiguration Wählen	tbd
S15	Abmelden.	tbd
S16	Admin UI - Anmeldung mit korrekten Daten	tbd
S17	Admin UI - Anmeldung mit ungültigen Daten	tbd
S18	Admin UI - Konfiguration Verwalten	tbd

8.2.3 Testplan Sprachsynthese

Szenario	Beschreibung	Resultat
S19	Benachrichtigung vorlesen - Sprachsynthese aktiviert und Benachrichtigung relevant	tbd
S20	Benachrichtigung nicht vorlesen - Sprachsynthese aktiviert und Benachrichtigung nicht relevant	tbd
S21	Lokale Einstellung - Sprachsynthese deaktiviert und Benachrichtigung relevant	tbd
S22	Lokale Einstellung - Sprachsynthese deaktiviert und Benachrichtigung nicht relevant	tbd
S23	Benachrichtigung verwalten - Relevanz Sprachsynthese kann im Admin UI aktiviert / deaktiviert werden	tbd
S24	Benachrichtigung empfangen - Änderung an Typ in Admin UI wird sofort angewendet	tbd

8.2.4 Testplan Gegensprechanlage

Szenario	Beschreibung	Resultat
S25	Gegensprechanlage Buttons nach Anmeldung anzeigen	tbd
S26	Verbindungsaufbau - Gegenüber ist Verfügbar	tbd
S27	Verbindungsaufbau - Gegenüber ist nicht Verfügbar	tbd
S28	Verbindungsaufbau - Gegenüber hat Gegensprechanlage deaktiviert	tbd
S29	Verbindungsaufbau - Benachrichtigungston	tbd
S30	Verbindungsaufbau - Automatische Annahme	tbd
S31	Unterhaltung 1:1 - Unterhaltung in Echtzeit möglich	tbd
S32	Unterhaltung 1:n - Unterhaltung in Echtzeit möglich	tbd
S33	Verbindungsaufbau 1:n	tbd
S34	Inbox - Vergangene Sprachverbindungen	tbd
S35	Inbox - Verpasste Sprachverbindungen	tbd
S36	Inbox - Abgelehnte Unterhaltungen	tbd
S37	Verbindung trennen durch Empfänger	tbd
S38	Verbindung durch Initiator	tbd
S39	Austreten aus Gruppenunterhaltung	tbd
S40	Konfiguration über Admin UI	tbd

8.2.5 Testplan Performance

Kriterium	Beschreibung	Resultat
P01	Zeit bis Benachrichtigung ankommt im Schnitt ¡ 5s	tbd
P02	Vorlesen von Benachrichtigung ¡ 5s nach Benachrichtigungston (ohne Cache)	tbd
P03	Vorlesen von Benachrichtigung ¡ 1s nach Benachrichtigungston (mit Cache)	tbd
P04	Verbindungsaufbau Sprachverbindung ¡5 s	tbd
P05	Verzögerung bei Sprachverbindung klein genug für normale Unterhaltungen	tbd
P06	Ressourcenverbrauch der Applikation bleibt über Zeit konstant	tbd

8.3 Fazit

In diesem Kapitel werden die zentralen Herausforderungen während der Projektarbeit und die Schlussfolgerungen die daraus gezogen werden können beschrieben.

Eine grosse Herausforderung in diesem Projekt, war die Konzipierung und Umsetzung des nativen Mobile CLients. Insbesondere die effiziente Anbindung an die Umsysteme Cloudservice und Messagingservice sowie die Integration von Sprachverbindungen mit WebRTC stellten eine Herausforderung dar. Die iOS Standardbibliothek sowie die iOS SDKs für WebRTC und Firebase Cloud Messaging und bieten alle Komponenten, welche für diese Integration notwendig sind. Die Herausforderung bestand darin, diese Komponenten effizient zu verwenden und Strukturen aufzubauen, welche die Integration in eine SwiftUI Applikation ermöglichen. Das Erarbeiten dieser Konzepte hat mehr Zeit in Anspruch genommen als erwartet und hat einen grösseren Teil des Konzepts in Anspruch genommen als erwartet. Dieser Mehraufwand hat sich schlussendlich aber bezahlt gemacht. Die Anbindungen an Umsysteme und Peer To Peer Verbindungen konnte in eigenen Komponenten gekapselt, welche effizient in SwiftUI eingebunden werden können.[8] Im Unterschied zur Entwicklung des Shared Platform Mobile Clients im Vorgängerprojekt, konnte der Aufwand hier mehrheitlich auf konzeptioneller Ebene gehalten werden. Die Anbindung der Schnittstellen und insbesondere die Verwendung von Gerätehardware und Betriebssystemfunktionen wie Pushbenachrichtigungen konnt deutlich einfacher umgesetzt werden. Es sind keine Probleme bezüglich Kompatibilität oder nicht unterstützten Funktionen aufgekommen.

Ich schliesse daraus, dass sich die native Mobile Entwicklung mit SwiftUI grundsätzlich besser für eine Praxisruf Applikation eignet als die Shared Platform Entwicklung. Um dies effizient zu machen, ist es allerdings unerlässlich, dass die Konzepte zur Anbindung von Umsystemen und direkten Verbindungen sauber erstellt werden. Die Verantwortlichkeit interne Komponenten muss klar definiert und der Aufbau effizient implementiert sein. Ist dies gegeben, kann am ende ein gutes Produkt stehen.

Das Erarbeiten der Konzepte für die Einbindung von WebRTC waren aus weiteren Gründen mühsam. Wie auch Firebase Cloud Messging (FCM) bietet WebRTC einen nativen iOS SDK. Im Unterschied zu FCM bietet WebRTC allerdings keine nennenswerte Entwicklerdokumentation. Dieses Risiko wurde bereits bei der Evaluation der Technologie[9] erkannt. WebRTC wurde trotzdem für dieses Projekt verwendet, da es Providerunabhängigkeit und maximale Flexibilität bei der Integration in das System bietet. Diese Vorteile konnten beim Projekt wirklich genutzt werden. Der Signalingservice ist komplett Providerunabhängig und kann bei einem beliebeingen Cloudprovider oder auf einem eigenen Server betrieben werden. Die Sprachverbindungen die im Mobile Client aufgebaut werden sind direkte Peer To Peer Verbindungen, auch dafür wird keine zusätzliche Instanz benötigt. Die mangelhafte Dokumentation hat die Umsetzung dieser Lösung allerdings deutlich erschwert. Es finden sich viele öffentlich zugängliche Referenzimplementierungen und einfache Anleitungen zur Integration von WebRTC in Applikationen. Diese sind in aller Regel aber sehr simpel gehalten. Sie beinhalten keine Mechanismen zum Verbindungsmanagement und keine saubere Integration in die Benutzeroberfläche. Die Komponenten aus dem WebRTC SDK werden kommentarlos verwendet. Dieses Problem wird ein Stück weit dadurch relativiert, dass WebRTC eine Open Source Technologie ist die von allen grossen Browsern unterstützt wird. Die Konzepte welche für den Verbindungsaufbau mit WebRTC verwendet werden, sind deshalb an vielen Orten beschrieben. Die Komponenten und Konzepte in WebRTC sind dabei Platformunabhängig dieselben. Dementsprechend konnten diese Resourcen verwendet werden, um das System zu versehen und auf die eigenen Anforderungen zugeschnitten umzusetzen. Schlussendlich konnte hier eine Lösung umgesetzt werden, die alle Anforderungen einer Gegensprechanlage im Praxisrufsysteme rfüllt.

Aus den Erfahrungen mit WebRTC im iOS Umfeld schliesse ich darauf, das WebRTC durchaus geeignet ist um ein Praxisrufsystem umzusetzen. Die Unabhängigkeit von Providern bringt grosse Flexibilität

[8] Siehe Kapitel 5
[9] Siehe Kapitel 4

und Unabhängigkeit mit sich. Gleichzeitig, muss aber betrachtet werden, dass WebRTC eine Open Source Technologie von Google ist. Es gibt keine Garantie wie lange WebRTC weiterentwickelt wird oder dass es mit zukünftigen iOS Versionen kompatibel bleibt. WebRTC selbst ist aber ein offener Standard und Google liefert lediglich die Implementation. Da es heute in allen grossen Browsern unterstützt ist, ist es wahrscheinlich dass es in jedem Fall von jemandem weiterentwickelt wird. Um sicherzustellen, dass ein Praxisrufsystem das WebRTC verwendet langfristig erfolgreich bleibt, muss eine entsprechende Ausstiegsstrategie definiert werden. Dies beinhaltet zeitnahe evaluation neuer iOS Releases um die Kompatibilität mit WebRTC sicherzustellen. Es beinhaltet weiter ein Konzept, wie WebRTC durch eine andere Technologie ersetzt werden kann. Im Rahmen dieser Projektarbeit konnte kein solches Konzept erstellt werden. Es wird empfohlen für die Weiterentwicklung von Praxisruf ein solches Konzept zu erstellen.

Covid war auch eine Challange. Im Methoden Teil wurde angedacht, dass scrum mässig zusammengesessen und getestet wird. Das hat aus zwei gründen nicht ganz wie erwartet funktioniert. Einerseits, ist der Konzept teil zu lang. Nicht länger als angedacht, aber halt doch lang. Meetings mussten grösstenteils remote statfinden. Das hat Demonstartionen und Absprachen deutlich erschwert. Anforderungen wurden am Anfang gemacht, das ist auch gut so. Persönlichere Meetings hätten aber vlt direkteres Feedback ermöglicht, so dass direkter auf Bedürfnisse hätte eingegangen werden können. Letztlich bin ich selbst am Covid erkrankt. Genau in der Zeit in der ich vorgenommen hatte, Zeit für das Projekt zu investieren. Das hat zu Verzögerungen geführt. Insgesamt trotzdem erreicht. Aber es könnte besser sein. Anforderungen waren als Minimum gedacht, mit raum für mehr.

Fazit: Puffer sind nötig. Es wurde Zeit für Polishing eingeplant aber nicht genug. Künftig: Puffer explizit als Puffer einbauen und nicht als Zeit in der man erwartet noch mehr machen zu können. Mehr Zeit für Testing Mehr Zeit für Polishing

Insgesamt bin ich mit dem Resultat dieser Arbeit sehr zufrieden. Ich bin sehr zufrieden mit der Systemarchitektur. Überzeugt, dass diese verwendet werden kann um ein gutes, kommerzielles Produkt zu erstellen. Ich bin weiter zufrieden mit dem Aufbau des Mobile Clients. Besonders da es mein erster ist. Besonders Anbindung umsysteme und integration in UI. Gleichzeitig hätte ich mir gwünscht weiter zu kommen. Es wurden gerade die minimalen Anforderungen unmgesetzt, die am Anfang definiert wurden. Eigentlich hätte ich mehr gewollt.

Unterm Strich: Ein guter Prototyp der als Basis für eine kommerzialisierung eines Cloudbasierten Praxisrufsystems dienen kann.

9 Schluss

Im Rahmen dieser Projektarbeit wurde Sprachübertragung und Sprachsynthese in ein cloudbasiertes Praxisrufsystem integriert. Die umgesetzte Lösung basiert auf dem Praxisrufsystem das im Rahmen des Projektes "Cloudbasiertes Praxisrufsystem" umgesetzt wurde.[2] Das erweiterte System besteht aus einer Mobilen Applikation, einem Cloudservice und einer Web-Applikation. 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 Funktionen und Anbindungen an Umsysteme auch in der neuen Applikation implementiert. Mit diesem Projekt neu konzipiert und umgesetzt wurden die Sprachsynthese für empfangene Benachrichtigung mit "AWS Polly " sowie die Integration einer Gegensprechanlage über Peer To Peer Verbindungen.

Das umgesetzte Praxisrufsystem kann zum Austausch von Informationen in einem Praxisumfeld verwendet werden. Als Endgeräte dienen dabei iOS Tablets. Über die Mobile Applikation des Systems ist es möglich Sprachverbindungen zu einem oder mehreren anderen Clients aufzubauen. Eingehende Sprachverbindungen werden automatisch angenommen. Das System unterstützt weiter das Versenden und Empfangen von Benachrichtigungen. Dies wird einerseits verwendet, um nicht erreichbare Empfänger über verpasste Sprachverbindungen zu informieren. Weiter bietet die Applikation Praxismitarbeitenden die Möglichkeit vorkonfigurierte Benachrichtigungen an andere Clients zu versenden. Der Inhalt von Benachrichtigungen kann dabei beim Empfang automatisch vorgelesen werden. Sowohl empfangene Benachrichtigungen als auch verpasste und vergangene Anrufe, werden gesammelt und in einer Inbox angezeigt. Empfangene 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. Praxisruf unterstützt in der aktuellen Version die Authentifzierung mittels Json Web Tokens. Ausstellung der Tokens wird dabei allerdings durch Praxisruf selbst gemacht. Es wird empfohlen vor der kommerziellen Nutzung einen externen Identity Provider anzubinden und Authentifizierung/Authorisierung nach OpenID Connect umzusetzen. Weiter ist Praxisruf heute nur beschränkt mandantenfähig. Praxismitarbeitende haben nur Zugriff auf Konfigurationen, welche dem verwendeten Benutzer zugewiesen sind. Praxisadministratoren können allerdings immer alle bekannten Konfigurationen über das Admin UI verwalten. Weiter wird bei der Auswertung der Konfiguration für das Zustellen von Benachrichtigungen und beim der Signalvermittlung für Sprachverbindungen keine Zuordnung an Benutzer oder Mandat geprüft. Um Praxisruf produktiv bei mehreren Kunden einzusetzen, muss es Mandantenfähigkeit implementiert werden. Letztlich sind heute einfache Mechanismen für das Wiederholen von Benachrichtigungen und Wiederaufbau von verlorenen Verbindungen implementiert. Für den produktiven Betrieb können und müssen diese allerdings noch erweitert werden. Insbesondere der Wiederaufbau von bestehenden Sprachverbindungen bei Verbindungsverlust ist für eine kommerzielle Nutzung unerlässlich.

Die grösste Gefahr für die produktive Nutzung von Praxisruf ist allerdings, das es bis heute nie in grösserem Umfang produktiv eingesetzt wurde. In der aktuellen Form sollte Praxisruf nicht im grossen Stil produktiv eingesetzt werden. Es bietet allerdings alle Funktionen, die ein Praxisrufsystem benötigt. Dementsprechend ist es möglich, das System in einem Pilotbetrieb einzusetzten. So kann Testfeedback von Benutzern eingeholt werden und es können Performance Metriken gesammelt werden. Diese können weitere Einblicke darauf geben, welche Teile des Cloudservices separat als skalierbare Microservices deployed werden sollen. Wird Praxisruf mit den Erkentnissen aus einem Pilotbetrieb ergänzt und die Funktionen Mandantenfähigkeit und OpenId Connect implementiert, kann es kommerziell und produktiv genutzt werden.

Insgesamt bin ich mit den Konzepten und Ergebnissen, die aus dieser Arbeit hervorgegangen sind zufrieden. Ich bin überzeugt, dass die erarbeiteten Konzepte und das umgesetzte System eine solide Grundlage für ein kommerziell erfolgreiches cloudbasiertes Praxisrufsystem bilden.

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] I. Amazon Webservices. (). Amazon Polly, Adresse: https://aws.amazon.com/polly/.

[4] Google Developers. (). 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] A. Inc. (). Swift, Adresse: https://developer.apple.com/swift/.

[7] ——, (). UIKit, Adresse: https://developer.apple.com/documentation/uikit/.

[8] ——, (). SwiftUI, Adresse: https://developer.apple.com/xcode/swiftui/.

[9] ——, (). Timer, Adresse: https://developer.apple.com/tutorials/swiftui/interfacing-with-uikit.

[10] firebase. (). Firebase iOS SDK, Adresse: https://github.com/firebase/firebase-ios-sdk.

[11] G. Developers. (). Set up a Firebase Cloud Messaging client app on Apple platforms, Adresse: https://firebase.google.com/docs/cloud-messaging/ios/client.

[12] A. Inc. (). AppDelegate, Adresse: https://developer.apple.com/documentation/uikit/uiapplicationdelegate.

[13] ——, (). URLRequest, Adresse: https://developer.apple.com/documentation/foundation/urlrequest.

[14] ——, (). Timer, Adresse: https://developer.apple.com/documentation/foundation/timer.

[15] ——, (). BGTaskScheduler, Adresse: https://developer.apple.com/documentation/backgroundtasks/bgtaskscheduler.

[16] ——, (). Speech Synthesis, Adresse: https://developer.apple.com/documentation/avfoundation/speech_synthesis.

[17] I. Amazon Webservices. (). Amazon Polly iOS Example, Adresse: https://docs.aws.amazon.com/polly/latest/dg/examples-ios.html.

[18] ——, (). Amazon Polly SDKs, Adresse: https://aws.amazon.com/polly/developers/.

[19] ——, (). 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] ——, (). Amazon Chime, Adresse: https://aws.amazon.com/chime/?chime-blog-posts.sort-by=item.additionalFields.createdDate&chime-blog-posts.sort-order=desc.

[22] ——, (). AWS Chime SDK for iOS, Adresse: https://github.com/aws/amazon-chime-sdk-ios.

[23] BlogGeek. (). WebRTC Mesh, Adresse: https://webrtcglossary.com/mesh/.

[24] ——, (). WebRTC MCU, Adresse: https://webrtcglossary.com/mesh/.

[25] C. P. M. Handley V. Jacboson. (). SDP: Session Description Protocol, Adresse: https://datatracker.ietf.org/doc/html/rfc4566.

[26] Mozilla. (). Introduction to WebRTC protocols, Adresse: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols.

[27] W3C. (). WebRTC 1.0: Real-Time Communication Between Browsers, Adresse: https://www.w3.org/TR/webrtc/.

[28] A. K. C. H. E. J. R. jdrosen.net. (). Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal, Adresse: https://datatracker.ietf.org/doc/html/rfc8445.

[29] A. Inc. (). URLSession, Adresse: https://developer.apple.com/documentation/foundation/urlsession.

[30] ——, (). Result, Adresse: https://developer.apple.com/documentation/swift/result.

[31] ——, (). Decodable, Adresse: https://developer.apple.com/documentation/swift/decodable.

[32] ——, (). URLSession.downloadTask, Adresse: https://developer.apple.com/documentation/foundation/urlsession/1411511-downloadtask.

Abbildungsverzeichnis

A Aufgabenstellung

Fachhochschule Nordwestschweiz

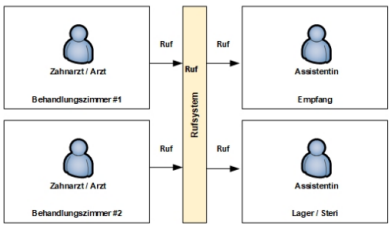
App
Web

21HS_IMVS38: Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem

			Priorität 1	Priorität 2
Betreuer:	Daniel Jossen	**Arbeitsumfang:**	P6 (360h pro Student)	---
		Teamgrösse:	Einzelarbeit	---
Sprachen:	Deutsch			

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 Signalübermittlung 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-Speach 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 Mutlicast 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. Der Quellcode ist für Berechtigte unter github.com einsehbar[10]. Berechtigungen können bei Joshua Villing angefordert werden.

[10] `https://github.com/users/jsvilling/projects/3`

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 wer
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 angezeig
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:	Verbindungsaufbau - Automatische Annahme
Given:	Empfang von Sprachverbindungen ist aktiiert
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:	Ale Ziele einer Sprachverbindung sind beim Signaling Service registriert
And:	Ale 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:	Kenn jedes Ziel mit dem Initiator Echtzeit kommunizieren.
Then:	Können die Ziele nicht direkt kommunizieren.

Scenario S33: Verbindungsaufbau 1:n

Given:	Ale Ziele einer Sprachverbindung sind beim Signaling Service registriert
And:	Ale 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:

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 Ehrlichkeitserklärung

≪Hiermit erkläre ich, die vorliegende Projektarbeit IP6 - Cloudbasiertes 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	01.03.2022

Unterschrift .....