Fachhochschule Nordwestschweiz
Hochschule für Technik

Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem

IP6 - Bachelor Thesis

14. Februar 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. 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.

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

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.”[1] [1]

Mit dem Projekt IP5 Cloudbasiertes Praxisrufsystem[2] wurde bereits eine mobile Applikation für Praxisruf umgesetzt. Mit dieser Applikation können bereits heute Benachrichtungen über Praxisruf versendet und empfangen werden. Um die Wartbarkeit und Hardware- sowie Betriebssystemkomaptibilität zu gewährleisten wurde im Fazit des Vorgängerprojekts empfohlen, die Applikation neu als native Applikation für iOS und Android zu schreiben.[2] Dieses Projekt beinhaltet damit die Migration des bestehenden Mobile Clients in eine native iOS Applikation sowie die Erweiterung um Sprachbezogene Funktionen.

Bei Projektstart bestehen zwei primäre Gefahren. Die erste Gefahr besteht in der unerfahrenheit der Projektteilnehmer mit Mobileentwicklung und inbesondere im Bezug Sprachkommunikation. Das Projektteam hat lediglich aus dem Vorgängerprojekt{ip5 Erfahrungen mit mobiler Entwicklung. Dies beinhaltet aber keine native iOS Entwicklung, wie sie für dieses Projekt notwendig ist. Weiter besteht keinerlei Erahrung mit der Verarbeitung und Übertragung von Sprachdaten und Telefonie. Die Gefahr für das Projekt besteht hier darin, dass die Evaluation und Umsetzung länger als erwartet gehen können oder das Konzepte nicht wie erwartet umgesetzt werden können. Dieses Risiko kann durch Recherche und Proof Of Concepts minimiert werden. Die zweite Gefahr besteht in der pandemischen Situation bei Projektstart im Herbst 2021. Es ist damit zu rechnen, dass viele Meetings nicht persönlich stattfinden können. Die Organisation und Kommunikation des Projektes wird auf die Einschränkungen der aktuellen Lage angepasst und von Anfang primär über digitale Werkzeuge organisiert. Präsentationen, Demonstrationen und Tests mit dem Kunden werden, wenn möglich vor Ort abgehalten. Ist dies nicht möglich werden auch Sie über Microsoft Teams abgehalten. Weiter ist es möglich, dass Projektteilnehmende Krankheitshalber ausfallen und sich das Projekt deshalb verzögert. Sollte dies der Fall werden müssen Umfang und Prioritäten im Projekt neu evaluiert und gegebenenfalls angepasst werden.

Die Umsetzung des Projekts erfolgt in vier Phasen. Zu Beginn werden die grundlegenden Anforderungen und der Umfang des zu entwickelnden Produkts definiert und festgehalten. Diese Anforderungen werden

[1] Ausgangslage, Ziele und Problemstellung im Originaltext der Aufgabenstellung

in Form von Meilensteinen erfasst und priorisiert. In der zweiten Phase werden Technologien evaluiert und ausgewählt, um die Anforderungen an das System umzusetzen. Es werden die Konzepte erstellt, wie das System umgesetzt wird. Die Konzepte müssen dabei nicht abschliessend definiert sein. Es müssen aber Grundabläufe und Schlüsselentscheideungen definiert werden. Wo möglich werden einfache Proof Of Concepts erstellt, um die gewählten Technologien und Konzepte zu validieren. In der dritten Phase werden die Anforderungen schliesslich umgesetzt. Für die Umsetzung wird zuerst ein nativer Mobile Client aufgesetzt und die Funktionalität des bestehenden Mobile Clients übernommen. Anschliessend wird die Sprachsynthese implementiert, da diese eng mit der bestehenden Funktionalität verknüpft ist. Als drittes wird schliesslich die Funktion der Gegensprechanlage umgesetzt. Zum Schluss der dritten Phase ist Zeit eingeplant um die Applikation eingehend zu testen, fehler zu beheben und andere Optimierungen vorzunehmen. Es werden Tests zusammen mit dem Kunden durchgeführt. Dieser kann Feedback geben und Erweiterungs- und Änderungswünsche angeben. Diese werdend gemeinsam priorisiert und soweit wie möglich in das Produkt miteingearbeitet. Der Projektbericht wird während des gesamten Projektes geschrieben. Die vierte Phase zum Schluss des Projektes ist zur vervollständigung und Korrektur des Projektberichtes reserviert.

Im nachfolgenden Hauptteil werden die erarbeiteten Konzepte und Resultate vorgestellt. Zuerst werden Vorgehensweise, Projektplan und die Organisation für das Projekt. Anschliessend werden die Anforderungen vorgestellt, welche für Praxisruf umgesetzt werden. Es wird darauf beschrieben, welche Technologien für die Umsetzung verwendet wurden und begründet, wieso diese Technologien gewählt wurden. Das Kapitel Konzept beschreibt das detaillierte technische Konzept für Funktionsweise und Architektur von Praxisruf. Darauf wird das umgesetzte System und Herausforderungen während der Umsetzung vorgestellt. Am Ende der Arbeit stehen ein Fazit und Schlusswort mit Empfehlungen für das weitere Vorgehen.

2 Vorgehensweise

2.1 Projektplan

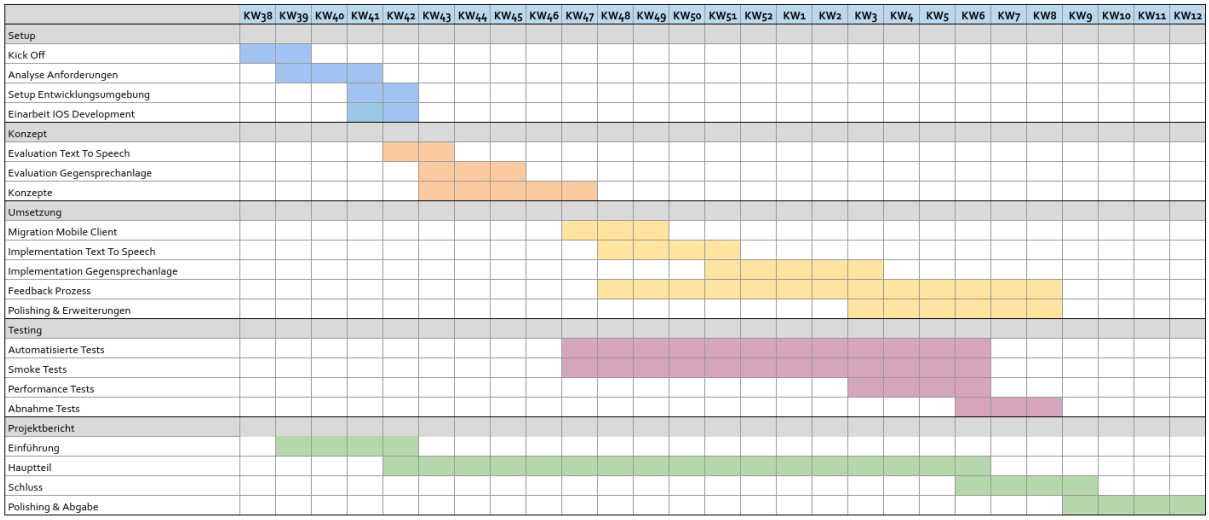


Abbildung 2.1: Projektplan

In der ersten Phase des Projektes (KW38-KW42) wird das Projekt aufgesetzt. Anforderungen und Umfang des Projektes werden zusammen mit dem Auftraggeber erarbeitet und anschliessend dokumentiert. Es wird ein Projektplan erstellt und die ersten Teile des Projektberichts werden aufgesetzt. Infrastruktur und Codebasis des Vorgängerprojektes wird übernommen und wo nötig für dieses Projekt angepasst. Für die Entwicklung des nativen Mobile Clients muss die Entwicklungsumgebung inklusive Apple Hardware aufgesetzt werden. Während dieser Zeit arbeitet sich das Projektteam bereits in die iOS Entwicklung ein um erste Erfahrungen zu sammeln und um zu validieren, dass die Entwicklungsumgebung vollständig und funktionsfähig ist. In der zweiten Phase (KW42-KW47) werden primär Konzepte erarbeitet und validiert. Dies beinhaltet das evaluieren von Technologien, das definieren und dokumentieren von Grundprozessen und Datenstrukturen. Die erstellten Konzepte sollen mit möglichst einfachen Proof Of Concepts validiert werden. Weiter wird dokumentiert, wie die bestehende Systemarchitektur erweitert wird um die neuen Funktionen zu ermöglichen. Dabei müssen insbesondere Skalierbarkeit, Erweiterbarkeit und Integrierbarkeit des Systems bewahrt werden. Die Umsetzung des Systems findet schliesslich in der dritten Pase (KW47-KW08) statt. Die festgehaltenen Anforderungen und Konzepte werden implementiert. Dabei wird as erstes der bestehende Mobile Client neu in eine native iOS Applikation migriert. Darauf wird der Client um die Sprachsynthesefunktion und letztlich um die Funktion Gegensprechanalge erweitert. Die erarbeiteten Konzepte werden in dieser Phase laufend überarbeitet und angepasst. Sobald die Applikation einen gewissen Reifegrad erreicht hat, soll der Kunde mittesten und Input geben können, welche Änderungen und Erweiterungen er wünscht. Diese werden soweit es der zeitliche Rahmen erlaubt mit eingearbeitet. Während der Umsetzung werden laufend automatisierte Tests geschrieben und Smoketests durchgeführt. Zum Ende der Umsetzungsphase sollen zudem Performance Metriken erfasst werden. Als letzter Test wird schliesslich ein Abnahmetest mit dem Kunden durchgeführt. Die vierte Phase (KW09-KW12) 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 und Gegensprechanalge (VOIP Kommuniation) ist abgeschlossen.
M04	**Konzepte** Die Konzepte für Systemarchitektur, Aufbau und Architektur der mobilen Applikation sowie Anpassungen an bestehenden Kompenenten im System sind abgeschlossen.
M05	**Migration betehender Funktionalität** Die Funktionen die im Mobile Client der Projektarbeit IP5 Cloudbasiertes Praxisrufsystem umgesetzt wurden, stehen in der neu entwickelten nativen IOS Applikation zur Verfügung.
M06	**Umsetzung** Alle Anforderungen zu der Funktion sind in der neu entwickelten nativen IOS Applikation umgesetzt.
M07	**Umsetzung Gegensprechanlage 1:1** Alle Anforderungen für die 1:1 Kommunikation über die Funktion Gegensprechanlage sind in der neu entwickelten nativen IOS Applikation umgesetzt.
M08	**Umsetzung Gegensprechanlage 1:n** Alle Anforderungen für die 1:1 Kommunikation über die Funktion Gegensprechanlage sind in der neu entwickelten nativen IOS Applikation 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.

3 Anforderungen

Es gibt drei Rollen von Stakeholdern, welche Anforderungen an Praxisruf stellen. Die meisten Benutzer des Systems fallen in die Rolle Praxismitarbeitende. Diese verwenden die mobile Applikation von Praxisruf, um in der Praxis miteinander zu kommunizieren. Neben der Rolle der Praxismitarbeitenden, arbeitet auch die Rolle des Praxisverantwortlichen mit dem Praxisrufsystem. Diese Benutzergruppe ist dafür verantwortlich, Praxisruf für Praxismitarbeitende zu konfigurieren. Als dritte Rolle hat zudem der Auftraggeber ein Interesse daran, dass gewisse Rahmenbedingungen gesetzt und eingehalten werden.Siehe Projektbericht Cloudbasiertes Praxisrufsystem [2].

Im folgenden Kapitel werden die Anforderungen dokumentiert, die bei Projektstart ermittelt wurden. Die Anforderungen werden dabei aus fachlicher Sicht mit User Stories festgehalten. Jede User Story beschreibt ein konkretes Bedürniss einer Stakeholdergruppe.

User Stories

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 wichtige 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 vergangene und verpasste Sprachverbindungen nachvollziehen können, damit ich mich zurückmelden kann.
U10	Als Praxismitarbeiter/in möchte ich, dass eingehende Sprachverbindungen aus anderen Praxiszimmern automatisch geöffnet werden damit ich meine Hände für besseres brauchen 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.

Praxisadministrator

Id	Anforderung
U13	Als Praxisadministrator möchte ich konfigurieren können, welche Benachrichtigungen dem Praxismitarbeitenden vorgelesen werden, damit nur relevante Benachrichtigungen vorgelesen werden.
U14	Als Praxisadministrator 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 Praxisadministrator möchte ich Benachrichtigungen, Clients und Benutzer wie zuvor konfigurieren können, damit ich das System weiterhin auf meine Praxis zuschneiden und bestehende Konfigurationen übernehmen kann.

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 bestehende 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 zu gewährleisten.
U19	Als Auftraggeber möchte ich, dass wo möglich der Betrieb von Serverseitigen Dienstleistungen über AWS betrieben wird, damit ich von bestehender Infrastruktur und Erfahrung profitieren kann.

Features

Aus den User Stories ergeben sich drei Features, welche mit dem Projekt P2P Sprachübertragung in Praxisrufsystemenümgesetzt werden müssen.

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

Diese Features dienen zur Aufteilung der Umsetzungsphase. F01 Migration des bestehenden Mobile Client bildet die Grundalge für die anderen Features und wird deshalb als erstes Umgesetzt. F02 Sprachsynthese ist eng mit der bestehenden Funktionalität verbunden. Es soll deshalb möglichst zeitnah nach F01 umgesetzt werden. F03 Gegensprechanlage ist das Kernstück der neuen Funktionalität. Dieses Feature kann erst umgesetzt werden, wenn die Grundlagenarbeit von F01 umgesetzt ist.

4 Technologie Evaluation

In diesem Kapitel wird evaluiert, mit welchen Technologien und Frameworks die Anforderungen für das Projekt umgesetzt werden. Dies beinhaltet die Migration des bestehenden Mobile Clients[2] (F01) sowie die Implementation der neuen Features ”Sprachsynthese” (F02) und ”Gegensprechanlage” (F03)[2].

4.1 Mobile Client

Mit dem Projekt ”IP5 Cloudbasiertes Praxisrufsystem” wurde bereits eine mobile Applikation für Praxisruf umgesetzt. Mit dieser Applikation können bereits heute Benachrichtungen über Praxisruf versendet und empfangen werden.[2] Die bestehende Applikation wurde mit Nativescript[3] als Multi-Platform Applikation gebaut. Um die Wartbarkeit und Hardware- sowie Betriebssystemkomaptibilität zu gewährleisten wurde im Fazit des Vorgängerprojekts empfohlen, die Applikation neu als native Applikation für iOS und Android zu schreiben.[2]

Mit diesem Projekt soll die Applikation dementsprechend neu als native iOS Applikation umgesetzt werden. 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.

Programmiersprache

Für die Entwicklung von nativen iOS Applikationen ist die Programmiersprache Swift Industriestandard.[4] Der native iOS Client für Praxisruf wird deshalb mit Swift implementiert.

Frameworks

Für die Umsetzung von iOS Applikationen stellt Apple die zwei Frameworks UIKit[5] und SwiftUI[6] zur Verfügung. UIKit ist das ältere der beiden Frameworks und ist seit iOS 2.0 verfügbar. Dementsprechend ist das Framework ausgereifter und bietet viele Funktionen zur Integration einer Applikation mit iOS.[5]

SwiftUI ist deutlich neuer und steht seit iOS 13.0 zur Verfügung. Auf der offiziellen Dokumentationsseite zu SwiftUI schreibt Apple SSwiftUI helps you build great-looking apps across all Apple platforms with the power of Swift — and as little code as possible.”[6] SwiftUI fokussiert sich auf eine declarative Syntax, welche es leichtgewichtiger als UIKit macht. Es bietet zudem ausgezeichnete Integration des Entwicklungsworkflows in die XCode Entwicklungsumgebung und viele Standardkomponenten wie Listenansichten, Formfelder und andere UIKomponenten. Dadurch wird es einfacher eine Benutzeroberfläche mit dem nativem Look und Feel einer nativen 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.[7] Es ist also grundsätzlich möglich, alles was mit UIKit umgesetzt werden kann auch mit SwiftUI umuzusetzen.

[2] Siehe Kapitel 3

Unterstützung Benachrichtigungen

Um Notifikationen über Praxisruf versenden und Empfangen zu können, muss Firebase Cloud Messaging in die Applikation integriert werden können. Firebase stellt für die Integration in iOS eine native Library zur Verfügung.[8] Die Integration von Firebase Cloud Messaging kann über Funktionen dieser Library implementiert werden. Dies beinhaltet die Registrierung bei Firebase Cloud Messaging sowie das Empfangen der Benachrichtigungen über Firebase. [9] Damit Push-Benachrichtigungen über das Betriebssystem angezeigt werden können, müssen die Empfangen Benachrichtigungen an das Betriebssystem übergeben werden. Mit AppDelegates ist es möglich sich in den Lifecycle des Betriebssystems einzuhängen[10]. Dadurch ist es auch möglich, Vorder- und Hintergrundbenachrichtigungen über das Benachrichtigungszenter von iOS anzuzeigen[9].

Die Firebase Cloud Messaging Library kann sowohl mit SwiftUI als auch mit UIKit verwendet werden. AppDelegates sind ein Konzept welches aus UIKit stammt[10]. SwiftUI Applikationen können ohne AppDelegates implementiert werden. UIKit Funktionen können allerdings nahtlos mit SwiftUI integriert werden.[7] Die Firebase Cloud Messaging Library für iOS ermöglicht es also, Benachrichtigungen von Praxisruf sowohl mit UIKit als auch mit SwiftUI umzusetzten.

Benachrichtigungen prüfen

Der Mobile Client soll den Benutzer regelmässig erinnern, wenn es ungelesene empfangene Benachrichtigungen gibt. Dazu muss regelmässig geprüft werden, ob es solche Benachrichtigungen gibt und gegebenenfalls ein Erinnerungston abgespielt werden. Die standard iOS Libraries von Apple bieten Mittel, mit welchen regelmässige Aufgaben angestossen werden können. Einerseits können über ”Timer” 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.[11] Da der bestehende Mobile Client dieselbe Einschränkung hat, könnte mit einem Timer trotzdem genau dasselbe Verahlten wie im bestehenden Client umgesetzt werden. Apple bietet allerdings auch Mittel, um Aufgaben im Hintergrund zu verarbeiten. Über die Klasse BGTaskScheduler können Aufgaben erfasst werden, die im Hintergrund ausgeführt werden.[12]

Die Umsetzung von der Erinnerungsfunktion kann also ohne zusätzliche Frameworks umgesetzt werden. Es besteht zudem die Möglichkeit den Umfang des bestehenden Clients zu erweitern und die Prüfungen auch im Hintergrund laufen zu lassen.

Entscheid

Der native iOS Mobile Client für Praxisruf wird mit Swift und SwiftUI umgesetzt. Der deklarative Syntax, ermöglicht es einfacher übersichtliche und lesbare Komponenten zu implementieren. Ausgezeichnete Integration in die XCode Entwicklungsumgebung und die zur Verfügung gestellten Standard Komponenten reduzieren den Entwicklungsaufwand weiter. Die Neuheit von SwiftUI bedeutet, dass möglicherweise gewisse Funktionen noch nicht mit purem SwiftUI umgesetzt werden können. Um diese Wahrscheinlichkeit zu minimieren, wird iOS15 als Zielplattform gewählt. Mit der Verwendung der neusten iOS Version als Zielplattform kann auch die neuste Version von SwiftUI und damit alle zur Verfügung stehenden Funktionen aus SwiftUI verwendet werden. Sollten trotzdem Funktionen benötigt werden, die SwiftUI nicht unterstützt, können diese dank der nahtlosen Integration mit UIKit implementiert und in Zukunft migriert werden.

4.2 Sprachsynthese

Praxisruf soll es neu Unterstützen, dass empfangene Benachrichtigungen vorgelesen werden.[3] Um dies zu ermöglichen muss eine Technologie integriert werden, die es erlaubt den Textinhalt einer Benachrichtigung in eine Sprachdatei zu konvertieren. Diese Sprachdatei muss Audio beinhalten, welches vom Mobile Client abgespielt werden kann.

Diese Integration kann mit den Standardbibliotheken für iOS oder durch die Anbindung eines externen Providers umgesetzt werden. Die Anbindung eines externen Provider kann entweder direkt im Mobile Client implementiert werden. Alternativ das Backend von Praxisruf[4] an den Provider angebunden werden und dem Mobile Client eine Schnittstelle bieten, um diese Daten abzufragen.

Apple Speech Synthesis

Die Standardbibliothek von für iOS unterstützt das Konvertieren von Text zu Sprache.[13] Sprachsynthese könnte dadurch ohne die Anbindung eines externen Providers umgesetzt werden. Durch die Verwendung Funktionalität die Apple selbst zur Verfügung stellt, ist zudem die Kompatibilität mit nativen iOS Clients garantiert. Gleichzeitig entsteht auch eine starke Bindung an Apple als Sprachsyntheseprovider. Sollte die Funktion in künftigen Versionen nicht mehr unterstützt werden, müsste die ganze Integration von Sprachsynthese neu implementiert werden. Weiter bringt diese Variante den Nachteil, dass der Mobile Client komplexer wird, weil er mit einer zusätzlichen Instanz kommunizieren muss. Letztlich hat diese Variante den Nachteil, dass dieselbe Funktionaltät für einen Android Client komplett neu entwickelt werden müsste.

Amazon Polly

Vom Auftraggeber ist explizit gewünscht, dass Infrastruktur und Services wo möglich über Amazon Webservices bestellt werden.[5] Mit dem Service ”Polly” bietet Amazon Webservices einen Service, welcher Text in Sprachdaten verwandeln kann.[14] Amazon Webservices stellt dazu Libraries für iOS[15] als auch für Java zur Verfügung[16]. Polly kann damit sowohl direkt in den nativen Mobile Client als auch in den Cloudservice integriert werden.

Polly von AWS Webservices bietet eine Library für die Intagration in iOS. [15] Mit dieser Library ist es möglich, die Anbindung des externen Sprachsyntheseproviders direkt im Mobile Client zu implementieren. Damit wird eine Lösung eingesetzt, die analog auch für zukünftige Android Clients eingesetzt werden kann. Die starke Bindung zu Apple wird durch diese Lösung ebenfalls aufgehoben. Sie wird allerdings durch eine starke bindung zu Amazon Webservices ersetzt. 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.

[3] F02 Sprachsynthese
[4] Cloudservice
[5] Siehe Kapitel 3 - F19

Alternativ ist es möglich, die Anbindung des Sprachsyntheseproviders im Cloudservice vorzunehmen. Dies hat einerseits den Vorteil, dass alle Clients eine einheitliche Schnittstelle haben können. Sowohl iOS als auch Android und WebClients können die Audiodaten über genau dieselbe Schnittstelle beziehen. Dies ermöglicht es auch den Provider in Zukunft auszutauschen ohne in den Clients etwas verändern zu müssen. Die Option hat zusaätzlich den Vorteil, dass Optimierungen, die nicht Client spezifisch sind getroffen werden können. So wäre es möglich einen externen File Storage anzubinden auf, welchem Audiodaten gespeichert werden. Dadurch müsste der Sprachsynthese Provider weniger oft angesprochen werden. Die Variante hat den Nachteil, dass der Cloudservice komplexer wird. Durch die Integration von Sprachsynthese muss aber mindestens eine Komponente im System ausgebaut werden. Wie stark die Komplexität des System wächsts, kann jedoch minimiert werden, indem die neue Funktionalität gekapselt wird. Sie kann so umgesetzt werden, dass sie unabhängig von restlichen System bleibt und alle nötigen Daten über die Schnittstelle der anderen Module bezieht.[6]

Entscheidung

Die Sprachsynthese wird durch die Anbindung des externen Providers AWS Polly umgesetzt. Der Cloudservice übernimmt die Kommunikation mit AWS Polly und bietet eine Schnittstelle, über welche Clients Audiodaten beziehen können.

Durch diesen Ansatz kann die Abhängigkeit zu einem spezifischen Provider minimiert werden und die Anbindung für alle Client Plattformen gleich gelöst werden. Dies macht diese Variante zukunftssicher und einfacher betreibbar. Der Einfluss von zusätzlicher Komplexität, die dieser Ansatz mit sich bringt, soll durch eine entsprechende Kapselung in der Systemarchitektur minimiert werden.

[6] Siehe Kapitel 5.1

4.3 Gegensprechanlage

Praxisruf soll um die Funktion einer Gegensprechanlage erweitert werden. Um dies zu ermöglichen, muss das System Sprachübertragung zwischen Mobile Clients in Echtzeit unterstützten. In diesem Kapitel werden die Technologien evaluiert, mit denen dies Umgesetzt werden kann.

4.3.1 Amazon Chime

Eine Möglichkeit um Sprachverbindungen in Praxisruf zu implementieren, ist die Integration einer bestehenden Business Kommunikationslösung, wie Webex oder Microsoft Teams. Amazon Webservices bietet für diesen Zweck "Amazon Chime" an. Amazon Chime ist ein Kommunkations Service welcher Funktionen für Meetings, Chats und Geschäftsanrufe ermöglicht.[17] Mit dem Amazon Chime SDK für iOS, ist es möglich Sprach- und Videoanrufe bis hin zu Screensharing in native iOS Applikationen zu integrieren.[18]

Die Integration eines solchen Dienstes 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 effizienteres Reagieren im Fehlerfall. Weiter fallen für Praxisruf selbst wenig bis keine Aufwände für den Betrieb der nötigen Telefonieinfrastruktur. Die Kosten, welche durch die Anbindung eines Providers anfallen, sind damit schlimmstenfalls gerechtfertigt und bestennfalls die günstigere Option, als die Infrastruktur selbst zu betreiben.

Die Funktionen für Chats, Videounterhaltungen, Meetings und Bildschirmübertragung sind für den Anwendungsfall "Gegensprechanlage in einem Praxisrufsystem" nicht relevant. Bei der Verwendung von Amazon Chime würde also nur ein kleines Subset der Möglichkeiten die der Service bietet verwendet. Der Nachteil daran ist, dass damit eine starke Bindung an den Amazon Chime SDK und die Abläufe, die Amazon Chime zum Verbindungsaufbau vorsieht.

4.3.2 WebRTC

WebRTC steht für Web Real-Time Communication. Es ist ein Open Source Projekt, welches Echtzeitkommunikation für mobile Applikationen und Browser Applikationen ermöglicht. Es unterstützt Video-, Sprach- und generische Daten, die zwischen Peers gesendet werden. Die Technologien hinter WebRTC sind als offener Webstandard implementiert und in allen gängigen Browsern als reguläre JavaScript-APIs verfügbar. Für native Clients wie Android- und iOS-Anwendungen steht eine Bibliothek mit derselben Funktionalität zur Verfügung.[19] WebRTC baut zur Kommunikation direkte Peer To Peer Verbindungen zwischen den Kommunikationspartnern auf. Um dies zu ermöglichen ist eine Signaling Instanz notwendig, welche Informationen die zum Verbindungsaufbau notwendig sind zwischen den Partnern vermittelt. WebRTC spezifiziert nicht, wie diese Signaling Instanz aussehen muss. Neben dieser Signaling Instanz ist keine Infrastruktur notwendig, um Daten auszutauschen. Sämtliche Sprachdaten werden direkt über die Peer To Peer Verbindungen zwischen Clients ausgetauscht.

Dies hat den Vorteil, dass die Infrastruktur schlank gehalten werden kann. Zudem kann der Signaling Service auf die eigenen Bedürfnisse zugeschnitten werden. Die Kanäle über die Signale ausgetauscht werden, können frei gewählt werden. Das Signaling im Kontext von Praxisruf kann damit im Cloudservice implementiert werden. Für die Benachrichtigung bei verpassten Anrufen können eigene Meachanismen eingebunden werden. Im Fall von Praxisruf, kann dafür der Notification Teil des Cloudservices verwendet werden. Diese Flexibilität bedeutet auch, dass der Signaling Service weitgehend unabhängig von der spezifischen Technologie implementiert werden kann.

Die Verwendung von WebRTC hat den Nachteil, dass kein fachlicher Support bei Verbindungsproblemen zur Verfügung steht. Dies wird dadurch relativiert, dass der eigene Signaling Service sehr einfach gehalten werden kann. Es wird weiter dadurch relativiert, dass der eigene Signaling Service ebenfalls bei einem Cloud Provider betrieben wird. Dafür können wiederum Supportvereinbarungen getroffen werden.

Die Verwendung von Peer To Peer Verbindungen ist effizient für Verbindungen bei 1:1 Unterhaltungen. Für 1:m Verbindungen müssen aber tatsächlich m Verbindungen aufgebaut werden. Bei grossen Gruppen, ist diese Art des Verbindungsaufbaus ineffizient. Grundsätzlich wäre es möglich, die Kommunikation über einen zentralen Peer zu bündeln.[20] Um dies zu ermöglichen ist aber wieder Infrastruktur nötig, über welche Sprachdaten übertragen werden können.

4.3.3 Entscheidung

Die Funktion Gegensprechanlage wir mit WebRTC umgesetzt. Dazu wird der Cloudservice um ein Modul erweitert, welches Signaling Informationen vermittelt. Im Mobile Client wird die WebRTC iOS Library integriert, um Peer To Peer Sprachverbindungen aufzubauen.

Durch die Verwendung von WebRTC, können Sprachverbindungen aufgebaut werden, ohne einen neuen externen Service zu integrieren. Weiter kann der Infrastruktur zur Signalvermittlung unabhängig von der gewählten Technologie umgesetzt werden. Die Verfügbaren Libraries für iOs, Android und Javscript bedeuten, dass künftige Web- und Android Clients leicht in das System integriert werden können. Support für die Verfügbarkeit des Signaling Services wird über die Verfügbarkeit des Cloud Providers sichergestellt.

5 Konzept

5.1 SystemArchitektur

Dieses Kapitel beschreibt die Erweiterungen und Anpassungen an der Systemarchitektur des bestehenden Praxisrufsystems. 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 optimieren.

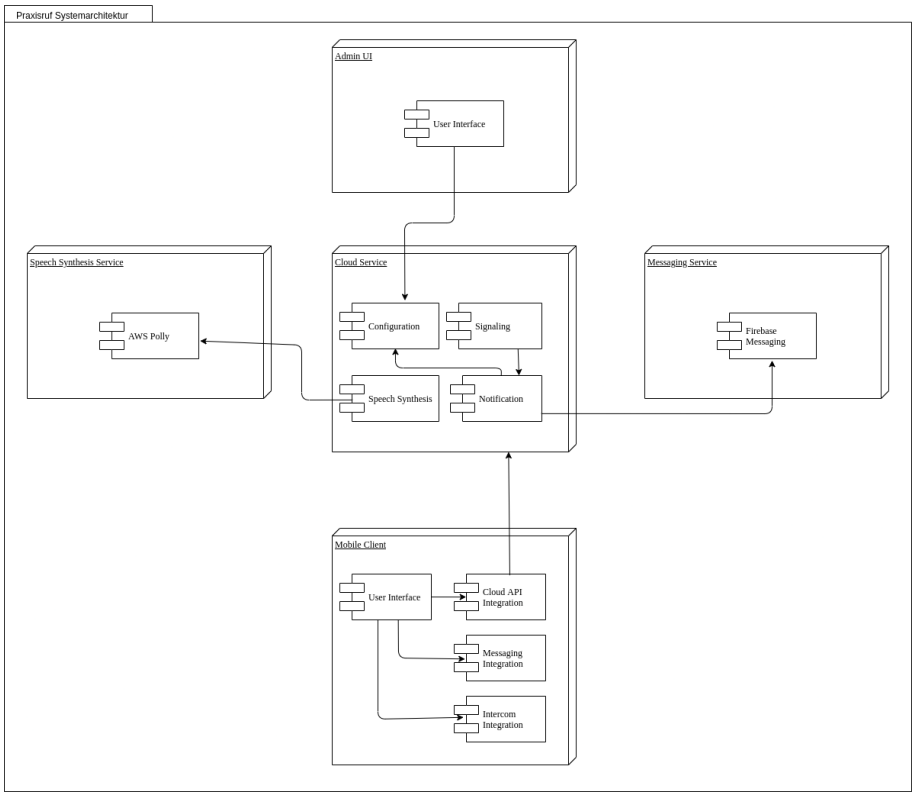


Abbildung 5.1: Systemarchitektur Praxisruf

5.1.1 Systemkomponenten

Dieses Kapitel gibt eine Übersicht zu den Systemkomponenten von Praxisruf. Dabei wird für jede Komponente beschrieben, welche Aufgaben ihr zufallen und welche Teile davon mit diesem Projekt angepasst werden.

Cloudservice

Der Cloudservice bildet das zentrale Backend des Praxisruf Systems. Der Cloudservice wird in die zwei Domänen Notification und Configuration aufgeteilt. Dabei ist die Domäne Notification für das Versenden von Benachrichtigungen und die Domäne Configuration für die Verwaltung und Auswertung der Konfigurationen verantwortlich. Zwischen den beiden Domänen besteht eine gerichtete Abhängigkeit. Die Domäne Notification benötigt zum Versenden von Benachrichtigungen Informationen aus der Configuration Domäne. Das Identifizieren der relevanten Empfänger geschieht in der Configuration Domäne. Dementsprechend benötigt die Notification Domäne beim Versenden die Information, an welche Empfänger die Benachrichtigung versendet werden soll.

Die Domänentrennung innerhalb des Cloudservices findet bis heute lediglich auf Package-Ebene statt. Mit diesem Projekt soll die Trennung einen Schritt weiter gehen. Die Applikation wird in mehrere Module aufgeteilt.[7] Diese können zu einem späteren Zeitpunkt in einzelne Microservices aufgeteilt werden.

Nachdem die Auftrennung in Module statgefunden hat, wird der Cloudservice um zwei Module erweitert. Das neue Modul Signaling ist für die Signalvermittlung zwischen Mobile Clients verantwortlich. Dies ist nötigt um Peer To Peer Sprachverbindungen zwischen Mobile Clients aufzubauen. Das Modul Signaling hat eine gerichtete Abhängigkeit zum Module Notification. Über das Notification Modul sollen Mobile Clients informiert werden, wenn ein Signal nicht versendet werden konnte, weil sie nicht mit dem Cloudservice verbunden waren. Das neue Modul Speech Synthesis dient als einheitliche Schnittstelle zu einem externen Seepch Synthesis Service. Dadurch kann auch wenn ein Android oder Web Client kommt, dieser genau gleich angebunden werden. Garantie, dass die Konfiguration und Funktionsweise dieselbe für alle Clients ist.

Mobile Client

Mit dem Vorgängerprojekt wurde bereits ein Mobile Client umgesetzt. Dieser wurde als Shared Platform Applikation mit Nativescript umgesetzt. Er bietet Praxismitarbeitenden die Möglichkeit Benachrichtigungen zu versenden und zu empfangen. Mit diesem Projekt wird der Mobile Client als native iOS Applikation komplett neugeschrieben. Der neue iOS Client muss sämtliche Funktionen des bestehenden Mobile Clients unterstützen. Zudem wird im neuen Client die Funktion der Gegensprechanlage und das Vorlesen von Benachrichtigungen über Sprachsynthese umgesetzt.

Admin UI

Das Admin UI ermöglicht dem Praxisadministrator, die Konfiguration des Systems zu verwalten. Die Konfiguration des Systems wird mit diesem Projekt erweitert. Einerseits können neu Buttons für die Gegensprechanalge konfiguriert werden. Andererseits kann eingestellt werden, welche Benachrichtigungen vorgelesen werden sollen. Beide Informationen müssen über das Admin UI verwaltet werden können.

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 anhnd der Konfiguration zu versenden. Die Anbindung des Cloudservices an den Messaging Service ist mit dem Vorgängerprojekt bereits umgesetzt und muss für dieses Projekt nicht angepasst werden. Der neu entwickelte native Mobile Client muss mit diesem Projekt an den Messaging Service angebunden werden, um Benachrichtigungen zu empfangen. Als Messaging Service wird Firebase Cloud Messaging verwendet.

Speech Synthesis Service

Um Sprach Synthese zu ermöglichen, wird ein externer Service angebunden. Dieser übernimmt die Konvertierung von Text aus Benachrichtigungen zu synthetisierter Sprache. Die Anbindung an den Speech Synthesis Service wird ausschliesslich im Cloudservice implementiert. Sämtliche andere Komponenten die Sprachdaten benötigen, fragen diese beim Cloudservice ab. Welcher die Abfrage an den Speech Synthesis Service übernimmt und die Resultate dem Anfrager zur Verfügung stellt. Als Speech Synthesis Service wird Amazon Polly verwendet.

[7] Siehe Kapitel 5.1.2

5.1.2 Modularisierung CloudService

Mit dem Projekt IP5 Cloudbasiertes Praxisrufsystem wurde der Cloudservice implementiert. Der Cloudservice trennt intern die beiden Domänen Notification und Configuration. Die Domäne Configuration ist für die Verwaltung und Auswertung der Konfiguration des Systems verantwortlich. Die Domäne Notification ist für das Versenden von Benachrichtigungen verantwortlich. Für das Versenden von Benachrichtigungen werden in der Notification Domäne Informationen aus der Configuration Domäne benötigt. Diese Informationen werden über eine REST-Schnittstelle abgefragt.

Die Trennung der Notification und Configuration Domänen, ermöglicht es die beiden Teile der Applikation in separaten Services zu betreiben. So können diese unabhängig voneinander betrieben und erweitert werden. Weiter wird es dadurch möglich, einzelnen Teilen der Applikation mehr Ressourcen zuzuteilen. Die Trennung der beiden Domänen in eigene Microservices wurde mit dem Projekt IP5 noch nicht vorgenommen. Die beiden Domänen wurden lediglich durch die Package Struktur innerhalb eines einzelnen monolithischen Services getrennt. Mit diesem Projekt soll die Trennung der Domänen einen Schritt weiter vorangetrieben werden.

Neu soll es pro Domäne ein Gradle Modul geben, welches sämtliche Domänenobjekte, Services und Schnittstellen der jeweiligen Domäne kapselt. Dadurch ist garantiert, dass die Domänen sauber voneinander getrennt sind. Sämtliche Kommunikation zwischen den Modulen muss über REST-Schnittstellen geschehen.

Um den Betrieb des Cloudservices für dieses Projekt möglichst simpel zu halten, werden die einzelnen Teile immer noch als monolithische Applikation gestartet. Dazu wird ein Modul **App** erstellt, welches alle Domänen Module kapselt und in einer Applikation startet. Um die Applikation zukünftig in einzelne Micro Services aufzutrennen, müssen die einzelnen Module um eine Klasse erweitert werden, welche nur dieses Modul als Applikation startet. 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. 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 Signalgvermittlung zwischen Mobile Clients.

5.2 Nativer Mobile Client

Mit IP5 wurde bereits ein Client umgesetzt. Dieser muss für IP6 migiert werden. Hier wird beschrieben, wie die bestehenden Anforderungen mit dem nativen client umgesetzt werden können.

5.2.1 Benutzeroberfläche

Die Ansichten zur Anmeldung und Zimmerauswahl werden analog zum bestehenden Mobile Client umgesetzt. Die Login Seite beinhaltet einen kurzen Willkommenstext und ein Logo für Praxisruf. Darunter findet sich ein einfaches Formular zur Eingabe von Benutzername und Passwort.

Abbildung 5.2: Mockup Login

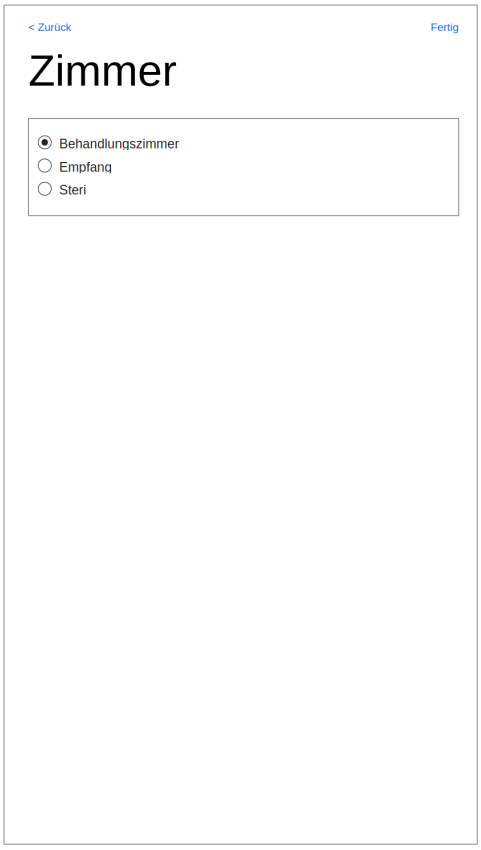


Abbildung 5.3: Mockup Zimmerwahl

Die Zimmerauswahl besteht aus einem Seitentitel und einem Panel indem das gewünschte Zimmer ausgewählt werden kann. In der Auswahl sind alle Zimmer zu sehen, welche dem Benutzer zur Verfügung stehen. In der Kopfzeile sind die Schaltflächen ”Zurück” und ”Fertig” zu sehen. Die Schaltfläche ”Zurück”, bricht die Anledung ab und führt zurück auf die Login Ansicht. 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.

In der Hauptansicht 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 Meldungen 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 Microfon und Lautsprecher des eigenen Gerätes stummgeschaltet werden. Weiter kann über der Anruf über den dritten Button beendet werden. Nach einem beendeten Anruf, wird zurück auf die Hauptansicht weitergeleitet. Für den Anrufern ist das immer der Home Bereich. Für den Empfänger entspricht es dem Bereich, der vor dem Empfang des Anrufes angezeigt wurde.

Abbildung 5.4: Mockup Home

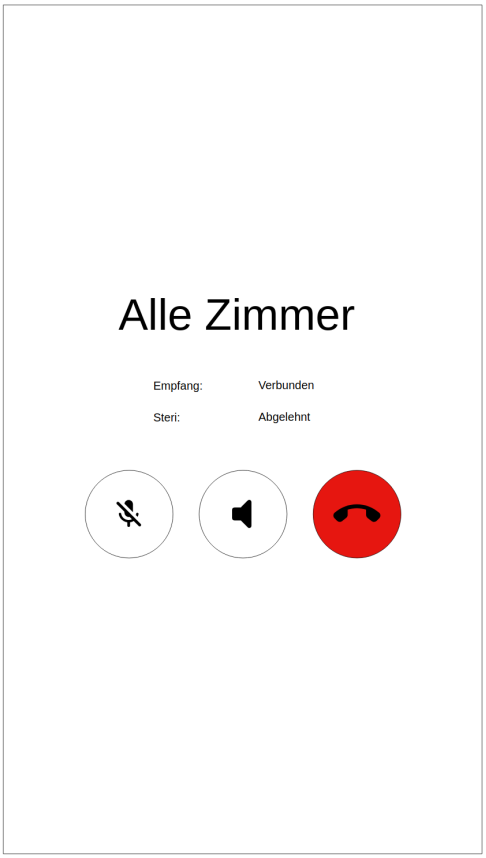


Abbildung 5.5: Mockup Aktiver Anruf

Der Bereich Inbox zeigt eine Liste der empfangenen Meldungen sowie der empfangenen und verpassten Anrufe. Für Meldungen wird der Titel der Benachrichtigung gefolgt vom Namen des Versenders in Klammern sowie der Haupttext der Meldung angezeigt. Für Anrufe wird der Name des Anrufers und ein Text der besagt ob es sich um einen empfangenen, verpassten oder abgelehnten Anruf handelt angezeigt. Einträge für Meldungen sowie verpasste und abgelehnte Anrufe müssen durch antippen quittiert werden. Solange unquittierte Meldungen oder Anrufe in der Inbox sind ertönt, wird im Abstand von 60 Sekunden eine Benachrichtigung angezeigt, die den Benutzer erinnert, die Inbox zu quittieren. Der Bereich Settings zeigt den Namen des aktuellen Benutzers und ausgewählten Zimmers. Über die Schaltfläche Abmelden, kann sich der Benutzer aus der Applikation abmelden. Die Schaltfläche Benachrichtigungen vorlesen ist standardmässig aktiviert. Wird die Option deaktiviert, werden Benachrichtigungen nie vorgelesen. Die Schaltfläche Anruge empfangen ist ebenfalls standardmässig aktiviert. Wird diese Option deakti-

viert, werden alle empfangenen Anrufe automatisch abgelehnt und stattdessen eine Benachrichtigung angezeigt. Ausgehende Anrufe können auch getätigt werden, wenn diese Option aktiviert ist.

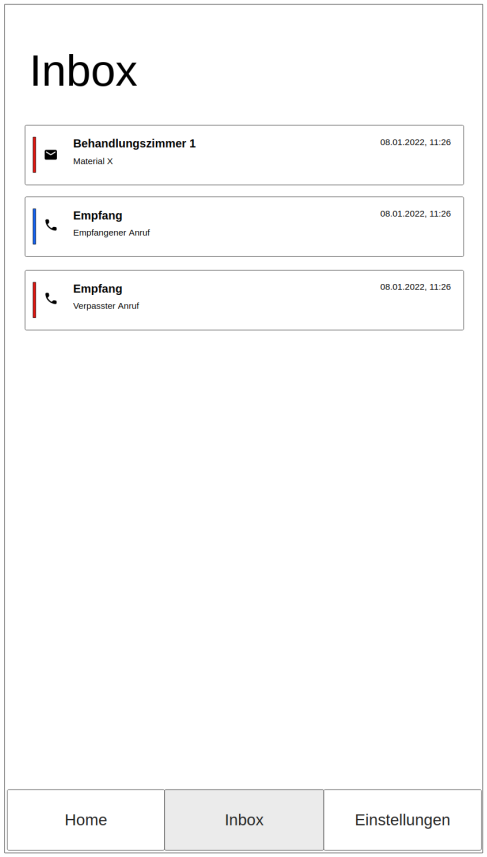


Abbildung 5.6: Mockup Inbox

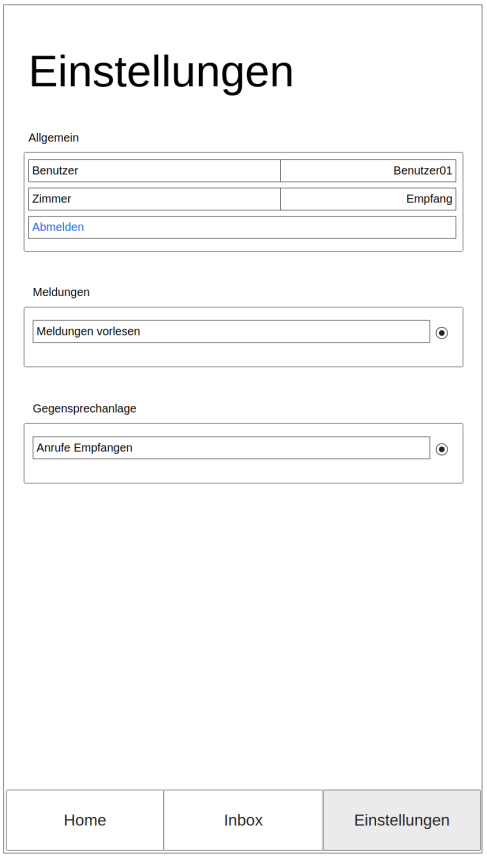


Abbildung 5.7: Mockup Einstellungen

5.2.2 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 REST Endpoints zur verfügung. In diesem Unterkapitel wird beschrieben, wie REST Aufrufe im nativen iOS Client integriert werden sollen.[8]

Die Basisbibliothek für iOS Entwicklung bietet die Klasse URLSession, über welche Netzwerkaufrufe getätigt werden können. Über URLSession.shared steht zudem eine Standard Instanz zur Verfügung, über welche Anfragen erstellt werden können.[21] Die Klasse UrlRequest ermöglicht es, Http-Request für eine URL mit Header und Body zu erstellen.[22] Um die Integration dieser Klassen in den Mobile Client zu vereinfach, wird ein zentraler Service mit dem Namen PraxisrufApi erstellt. Dieser kapselt das Erstellen, Befüllen und Absetzten der nötigen UrlRequest Instanzen und bietet öffentliche für die Http Verben Get, Post und Delete an.

Der Basis des Api Service lädt die Basis URL für die Praxisruf API us der Umgebungskonfiguration. Die http Methoden auf dem Basis Service nehmen eine subUrl für den Aufruf entgegen. Diese wird der Basis Url angehängt. Post und Put nehmen zudem einen Parameter data für den Inhalt des Request Bodies

[8] Die Anbindung der Domäne Signaling findet nicht über REST statt und ist im Kapitel 5.2.4 beschrieben.

entgegen. Aus diesen Informationen kann der Http Request gebaut und versendet werden. Als letzten Parameter nehmen alle Methoden einen Parameter mit dem Namen completion entgegen. Dabei handelt es sich um ein Callback, welches beim Erfolg oder Fehlschlagen der Anfrange an die API ausgeführt wird.

```
1  typealias CompletionHandler<T> = (Result<T, PraxisrufApiError>) -> Void
2
3  class PraxisrufApi {
4
5      static let httpBaseUrlValue = ...
6
7      func get<T>(_ subUrl: String,
8                  completion: @escaping CompletionHandler<T>) where T : Decodable {
9
10         http(subUrl, completion: completion)
11     }
12
13     func post<T>(_ subUrl: String,
14                  body: Data? = nil,
15                  completion: @escaping CompletionHandler<T>) where T : Decodable {
16
17         http(subUrl, method: "POST", body: body, completion: completion)
18     }
19
20     func delete<T>(_ subUrl: String,
21                    completion: @escaping CompletionHandler<T>) where T : Decodable {
22
23         http(subUrl, method: "DELETE", completion: completion)
24     }
25
26     func download(_ subUrl: String,
27                   completion: @escaping CompletionHandler<T>) { ... }
28
29     func http<T>(_ subUrl: String,
30                  method: String = "GET",
31                  body: Data? = nil,
32                  completion: @escaping CompletionHandler<T>) where T : Decodable {
       ... }
33
34 }
```

Listing 1: PraxisrufApi.swift

Als Input Paramter dieses Callbacks wird immer der Typ Result¡T, PraxisrufApiError¿ verwendet. Bei Result handelt es sich um einen Wrapper Type der entweder als Erfolg oder Fehler instanziert werden kann. Im Fehlerfall wird der Wrapper mit einem PraxisrufApiError Objekt instanziert. Im Erfolgsfall wird der Wrapper mit einem Object vom Typ T instanziert. Der Type T ist generisch und muss dem Decodable Protocol entsprechen. Decodable Instanzen können von einer JSON Spring Repräsentation in ein Swift Object konvertiert werden. So kann das Objekt im Erfolgsfall aus der Response generiert und an dass Callback übergeben werden. Dadurch kann die Konvertierung generisch in der Basisklasse behandelt werden. Innerhalb des Callbacks kann geprüft werden ob der Request erfolgreich war oder nicht und anhand des Inhalts die entsprechende Logik ausgeführt werden. Dieser Ansatz ermöglicht es im Api Service ausschliesslich den API Call abzuhandeln. 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 authorisiert. Dazu lädt der Service die hinterlegten Credentials aus dem KeyStore von iOS und generiert einen entsprechenden Authorization Header. Ist kein Token vorhanden wird der CompletionHandler mit einer entsprechenden Fehlermeldung aufgerufen.

Mit dieser Lösung steht ein Service zur verfügung, über welchen REST Calls einfach gemacht 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 jeder Call den Bedürfnissen des Aufrufers entsprechend abgehandelt werden. Die Extension Klassen mit domänenspezifischen, sprechenden Methoden machen die verfügbaren Aufrufe übersichtlich. Die API wird immer einheitlich über den generischen Service angesprochen. Die Integration in den rest der Applikation über sprechende Namen führt dabei zu lesbarem und übersichtlicherem Code.

Pro Domäne die angesprochen wird, wird eine Extionsionklasse erstellt. Diese bietet Methoden mit sprechenden Namen für die angesprochene Funktionalität. Die Methoden darin rufen die get, post, put und delete Methoden des API Service mit den entsprechenden Parametern auf.

Der Service zur API Integration wird nicht direkt in den View Komponenten verwendet. Es wird pro Domäne ein weiterer Service geschrieben, welche den Aufruf des API Services kapselt. Dieser Service bietet Methoden, über welche der API Service angesprochen werden kann. Die Methoden des Integration Services nehmen die Informationen, welche aus der Benutzeroberfläche Stammen als Parameter entgegen. Sind Daten aus der aktuell aktiven Client COnfiguration für den Request nötig, werden diese im Integration Service aus den UserDefaults geladen. Die Integration Services werden als ObservableObjects implementiert. Die Resultate der API Calls werden im Service als Instanzvariable mit @Published gesetzt. Die View, welche einen Integration Service verwendet, kann Bindings auf diese @Published Resultate setzten.

5.2.3 Anbindung Messaging Service

Als Messaging Service wird Firebase Cloud Messaging verwendet. Firebase bietet eine native library mit welcher Firebase Cloud Messaging in iOS Clients integriert werden kann. 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 Betriebsystem notwendig ist. Diese Integration kann bis heute nur über AppDelegates aus der UIKit Welt umgesetzt werden. SwiftUI Applikationen können oft ohne AppDelegates implementiert werden. Sobald aber Integration mit dem Betriebsystem notwendig ist, müssen AppDelegates verwendet werden. Dazu können AppDelegates bei der Initialisierung der Applikation registriert werden. Zur Anbindung von Firebase Cloud Messaging an den Mobile Client wird dementsprechend ein AppDelegate implementiert. Die Logik des AppDelegates soll dabei auf das minimal nötige reduziert werden. Der AppDelegate selbst ist für die direkte Kommunikation mit dem 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 MessagingServices im AppDelegate zu kapseln. Was den wechsel auf einen Anderen Messaging Service in Zukfunft vereinfacht. Weiter sorgt es dafür, dass die Fachlogik vollständig im SwiftUI Teil implementiert werden kann. Der AppDelegate beinhaltet lediglich die Teile, welche aus technischen Gründen nicht mit SwiftUI umgesetzt werden können.

Die Implementation des AppDelegate muss damit folgende Anforderungen erfüllen. 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 App Delegate 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 Appliation 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 weitern Verarbeitung übergeben werden.

5.2.4 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 Praxismitarbeiter darauf aufmerksam zu machen. Diese Prüfung findet bisher nur statt, wenn die Applikation in Vordergrund aktiv ist und nicht, wenn sie minimiert ist. Für den nativen Mobile Client soll die Funktion analog umgesetzt werden. Um dies zu ermöglichen sind zwei Komponenten möglich. Erstens, wird InboxService 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.[11] 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 als erstes nicht nur in Momory gehalten, sondern auf dem Gerät persistiert werden. So stehen die Daten auch zur Verfügung, wenn die Applikation nicht gestartet ist. Weiter muss ein mit Hintergrundtask[12] implementiert und registriert werden, welcher die persistierten Daten lädt und die darauf die Prüfung des InboxReminderService ausführt.

5.2.5 Security

Um die sichere Übertragung von Daten zu gewährleisten, werden alle Daten zwischen den Services über verschlüsselte Verbindungen ausgetauscht. HTTP Anfragen an de CloudService erfolgen ausschliesslich über HTTPS. Der Aufrufer für alle Abfragen muss mit einem entsprechenden JWT Token authentifiziert sein. Der Austausch von Daten über Websockets erfolgt ausschliesslich über Secure WebSockets (WSS). Für den Aufbau von Websocketverbindungen, muss der Benutzer ebenfalls authentifiziert sein. Dazu muss der Http-Handshake Request um die Websocketverbindung zu öffnen mit einem entpsrechend gültigem JWT Token authentifiziert werden.

Langfristig sollen Authentifizierung und Authorisierung in Praxisruf mit OAuth und OpenId-Connect umgesetzt werden. Diese Integration wurde mit dem Projekt IP5 Cloudbasiertes Praxisrufsystem noch nicht umgesetzt und fällt für dieses Projekt out of Scope. Die Authentifizierung des Mobile Clients wird im neuen nativen Client analog zum bestehenden Shared-Platform-Client umgesetzt. Um den Mobile Client zu verwenden, wird der Benutzer über Basic Authentication authentifiziert. Er erhält als Antwort auf diese Registrierung ein JWT Token welches ihn für alle weiteren Anfragen an den Cloud Service berechtigt. Sowohl die Credentials für die Basic Authentication als auch das JWT Token werden im Keystore von iOS gespeichert. Das JWT Token wird regelmässig erneuert, indem die Basic Authentication mit den gespeicherten Credentials wiederholt wird.

5.3 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 von Sprachdaten. Die Integration von notwendigen Schnittstellen im Mobile Client wird im Kapitel 5.2 beschrieben.

5.3.1 Konfiguration

Nach Use Case U03[9] muss der Praxismitarbeitende alle Sprachbenachrichtigungen stummschalten können. Dazu wird der Mobile Client um eine Settings Ansicht erweitert. Auf dieser Ansicht, kann der Benutzer auswählen, ob Benachrichtigungen vorgelesen werden sollen. Ist die Option deaktiviert, werden Benachrichtigungen nie vorgelesen.

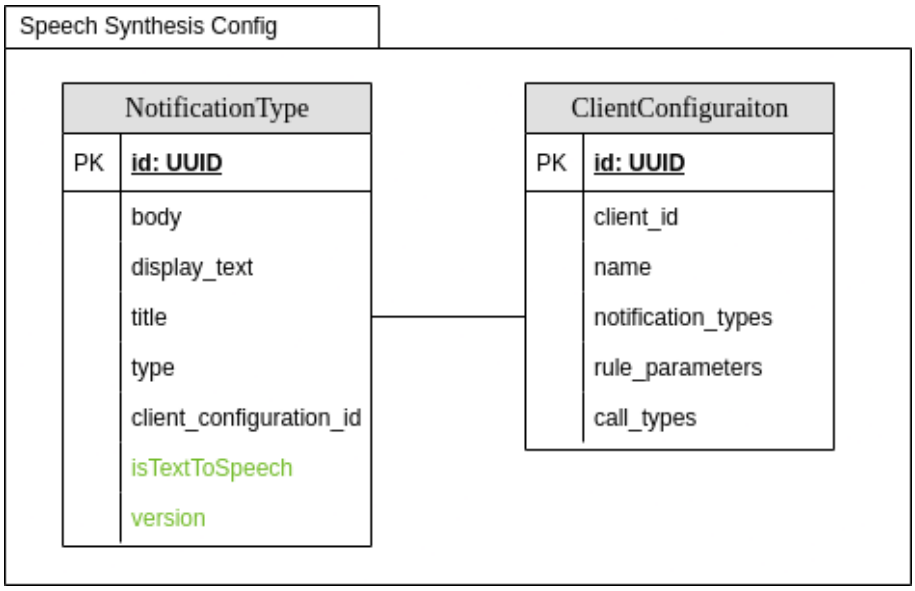


Abbildung 5.8: ERD Ausschnitt - Konfiguration Sprachsynthese

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 auf TRUE gesetzt haben vorgelesen.

Neben dem Feld isTextToSpeech, wird die NotificationType Entity um ein weiteres Feld version erweitert. Dieses Version Feld beinhaltet eine Ganzzahl welche mit jeder Änderung inkrementiert wird. Das Version Feld wird ebenfalls beim Versenden von Benachrichtigungen mitgesendet und kann auf Client Seite zur Implementierung eines Caches verwendet werden.

[9] Siehe Kapitel 3

5.3.2 Integration Sprachsynthese Provider

Die Anbindung des Sprachsynthese Providers erfolgt zentral im Cloudservice. Mobile Clients sprechen den Sprachsynthese Provider nie direkt an. Sie kommunizieren stattdessen mit dem Cloudservice, welcher Anfragen an den Sprachsynthese Provider ausführt und die Resultate an den Mobile Client weiterleitet. Dazu wird der Cloudservice um ein Modul mit dem Namen Speech Synthesis erweitert. Das Speech-Modul darf gleich wie die Module Configuration und Notification keine direkten Abhängigkeiten auf andere Cloud Service Module haben. Sämtliche notwendige Kommunikation zwischen den Modulen findet über Rest-Schnittstellen statt. Diese Unabhängigkeit ermöglicht es, das Modul in Zukunft einfach aus dem Cloudservice auszubauen und als eigenständigen Micro-Service zu betreiben.

Wie in Kapitel 4.2 beschrieben, wird AWS Polly als Sprachsynthese Provider für dieses Projekt verwendet. Demenstprechend, muss die Anbindung im Cludservice für AWS Polly implementiert werden. Die Abhängigkeit an einen spezifischen Provider soll dabei soweit wie möglich minimiert werden. So kann bei Bedarf einfacher auf einen anderen Provicer 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 als Parameter entgegennimmt. Der Parameter entsprocht dabei der technischen Identifikation des zu synthetisierenden Benachrichtigungstypes. Die InputStreamResource muss die snythetisierten Sprachdaten enthalten.

```
1  /**
2   * Contracts for getting speech synthesis data from a speech synthesis provider.
3   */
4  public interface SpeechSynthesisService {
5
6      /**
7       * Requests the NotificationType with the given id from the configuration
8       * and then sends a request to a speech synthesis provider.
9       *
10      * The result from the provider is written into a InputStreamResource
11      * containing MP3 audio data.
12      */
13     InputStreamResource synthesize(UUID id);
14
15 }
```

Listing 2: SpeechSynthesisService.java

Dieses Interface kann als Abhängigkeit im Endpunkt, mit welchem Mobile Clients kommunizieren verwendet werden. Um einen spezifischen Sprachsynthese Provider zu unterstützen, reicht es aus, dieses Interface zu implementieren und eine entsprechende Instanz zur Verfügung zu stellen.

Für dieses Projekt bedeutet dies, dass ein SprachSyntheseProviderService für AWS Polly implementiert werden. AWS Polly bietet einen Java SDK, welcher die Anbindung ermöglicht. Dieser SDK bietet alle Klassen die für die Anbindung an AWS Polly nötig sind und wird in der Implementierung des SprachSyntheseProviderService verwendet, um den Service anzubinden.

Konkret benötigt es für die Anbindung von AWS Polly drei Komponenten. Als Erstes muss ein AWSStaticCredentialsProvider zu Verfügung gestellt werden. Dieser liefert die Credentials, welche das System berechtigen, Anfragen an AWS Polly zu senden. Als zweites muss eine Voice konfiguriert werden. Diese definiert Sprache und Stimme, welche für die Synthetisierung der Sprachdaten verwendet wird. Letztlich muss ein 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 und die konfigurierte, Voice um die Daten entsprechend zu synthetisieren.

Da im Cloudservice Spring Boot verwendet wird, können diese Komponenten in einer Spring Konfigurationsklasse konfiguriert und instanziert werden. Im konkreten Service können diese dann als Abhängigkeiten angegeben werden und stehen über die Spring Dependency Injection zur Verfügung. Die Properties welche für die 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 im application.yml definiert und mit dem Quellcode des Projektes verwaltet. Als Credentials für die Anbindung dienen die zwei Schlüssel AccessKey und SecretKey.[10] Diese werden nicht direkt im application.yml abgelegt. Für sämtliche Credentiels wird im application.yml ein Platzhalter definiert, welcher die Credentials aus entsprechend benannten Umgebunsvariablen lädt. Die Zugangsdaten müssen damit nicht mit dem Quellcode verwaltet werden und können einfach ausgetauscht werden.

```
1  @Configuration
2  @ConfigurationProperties(prefix = "praxis-intercom.aws")
3  public class AwsConfiguration {
4
5      private String accessKey, secretKey, region, language;
6
7      @Bean
8      public AmazonPollyClient amazonPollyClient() {
9          ...
10     }
11
12     @Bean
13     public Voice voice(AmazonPollyClient polly) {
14         ...
15     }
16
17     private AWSStaticCredentialsProvider credentialsProvider() {
18         ...
19     }
20 }
```

Listing 3: AwsConfiguration.java

In der Implementation des Services kann nun über den injezierten AmazonPollyClient eine Abfrage an den Speech Synthesis Service gesendet werden.

[10] quote here

5.3.3 Schnittstelle Cloud Service

Das Speech-Modul stellt einen Endpoint zur Verfügung über den die Sprachdaten zu einer Benachrichtigung abgefragt werden können. Der Endpoint bietet dabei nicht die möglichkeit generische Daten in Sprachdaten zu verwandeln. Stattdessen erlaubt er es Sprachdaten für die aktuelste Version von bekannten Benachrichtigungstypen zu beziehen. Der einzige Parameter für diese Anfragen ist die technische Identifikation des Benachrichtigungstyps (NotificationType) der relevanten Benachrichtigung. Anhand dieses Identifikators können die Informationen des Benachrichtigugnstyps über die REST-Schnittstelle des Configuration-Modul angefragt werden. Dadurch können Änderungen an einem Benachrichtigungstyp direkt angewendet werden, ohne dass die Informationen auf dem Mobile Client aktualisiert werden müssen. Die geladenen Daten können dann verwendet werden um eine Anfrage an den Sprachsynthese Provier zu senden. Die vom Provider gelieferten Sprachdaten können anschliessend als Resultat an den Client zurügegeben 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. Auf Client Seite, kann dieser Endpunkt so als Download angesprechen werden.

```
1  @RestController
2  @RequestMapping("/api/speech")
3  @Api(tags = "Speech Synthesis")
4  @AllArgsConstructor
5  public class SpeechSynthesisController {
6
7      private final SpeechSynthesisService service;
8
9      @GetMapping(path = "/{id}", produces = "audio/mp3")
10     public ResponseEntity synthesizeTestAudio(@PathVariable("id") UUID id) {
11         InputStreamResource inputStreamResource = service.synthesize(id);
12         return new ResponseEntity(inputStreamResource, HttpStatus.OK);
13     }
14
15 }
```

Listing 4: SpeechSynthesisController.java

5.3.4 Integration Mobile Client

Für die Integration des SpeechSynthesis Endpunkts aus dem CloudService muss die in Kapitel 5.2 beschriebene Klasse PraxisrufApi erweitert werden. Neben dem Abfragen von JSON Daten über HTTP Schnittstellen, muss diese für die Sprachsynthese auch das herunterladen von Dateien unterstützten. Dies kann URLSession aus der iOS Standardbibliothek bietet dafür mit URLSession.downloadTask die Möglichkeit Inhalte von einer URL herunterzuladen.[23] Der Service PraxisrufApi wird dementsprechend um eine Methode download ergänzt. Diese ist dafür verantwortlich, eine Anfrage für den Download mit Credentials aus dem iOS Keystore zu ergänzen und die Abfrage zu versenden und die Resultate oder Fehler an den mitgegebenen CompletionHandler zu übergeben.[11] Heruntergaladene Dateien werden vom 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.

```
1 extension PraxisrufApi {
2       func download(_ subUrl: String, completion: @escaping CompletionHandler<T>) {
3         ...
4       }
5 }
```

Listing 5: PraxisrufApi+Download.swift

Die Sprachsynthese für Benachrichtigungen, soll automatisch ausgeführt werden sobald eine relevante Benachrichtigung empfangen wurde. Die Benachrichtigung wird im AppDelegate empfangen, aufbereitet und an die Applikation übergeben. Die aufberetieten Daten beinhalten, die Information, ob die Benachrichtigung für die Sprachsynthese relevant ist. Dies kann dem isTextToSpeech Flags in der Benachrichtigung entnommen werden. Für die Verarbeitung der Sprachdaten, wird ein eigener Service erstellt. Dieser SpeechSynthesisService verwendet die Downloadfunktion aus der PraxisrufApi Basisklasse, um Sprachdaten vom Cloudservice abzufragen. Wurden die Daten erfolgreich geladen, kopiert er die heruntergeladenen Daten aus dem temporären Downloadverzeichnis in ein permanentes Verzeichnis. Die Datei wird dabei unter dem Namen ”ID NotificationType”.”version” gespeichert. Anschliessend werden die Inhalte der Datei als Audio wiedergegeben. Die Namenskonvention für die gespeicherten Dateien, erlaubt es ein einfaches Cache zu implementieren. 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 versendet und die Datei wird direkt abgespielt. Die Id des NotificationTypes im Dateinamen sorgt dafür, dass pro Benachrichtigungstyp nur einmal Sprachdaten abgefragt werden müssen. Die Version des NotificationTypes im Dateinamen sorgt dafür, dass die Daten bei einer Änderung am Benachrichtigungstyp neu geladen werden. Da der Inhalt einer Benachrichtigung im Cloudservice abgefüllt wird, bedeuet dies das Benachrichtigungen immer in der neusten Version angezeigt werden. Durch die beschriebene Implementierung des Chaches ist sichergestellt, das auch immer diese Version für die Sprachsynthese verwendet wird.

[11] Vgl. Kapitel 5.2

5.3.5 Laufzeitsicht

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. Anschlissend erstellt er für jeden Empfänger eine Benachrichtigung und versendet diese über den Messagingservice.[2][12] Der Messagingservice stellt die Benachrichtigungen an die jeweiligen Empfänger zu. Im Folgenden wird der Ablauf vom Empfang der Benachrichtigung im Mobile Client bis hin zur Ausgabe der Sprachsynthese beschrieben.

Benachrichtigungen werden im Mobile Client durch den AppDelegate empfangen. Dieser beinhaltet die Anbindung an den Messaging Service und kann dementsprechend Benachrichtigungen vom Messaging Service entgegennehmen. Die Empfangene Benachrichtigung entspricht dem Format das der Messaging Service definiert. Im AppDelegate werden die Informationen aus diesem Format herausgelesen und in das interne Model der Mobile Client Applikation überführt. Anschliessend wird die Benachrichtigung an den CompletionHandler von iOS übergeben. Dadurch wird der Benachrichtigungston abgespielt und die Benachrichtigung als Push Benachrichtigung angezeigt. Daraufhin wird die Benachrichtigung im internen Model einem NotificationService übergeben. Dieser fügt die empfangene Benachrichtigung in die Inbox ein. Ab diesem Moment ist die Benachrichtigung in der Inbox des Mobile Clients ersichtlich.

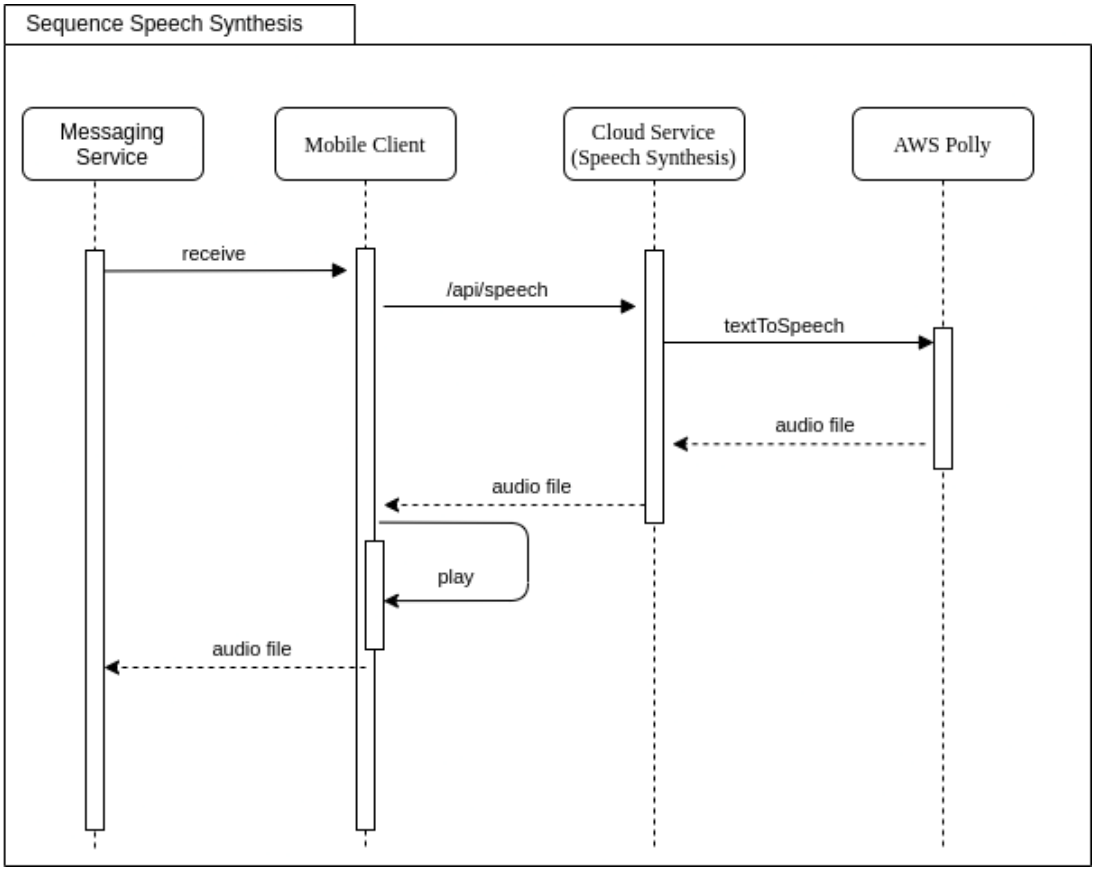


Abbildung 5.9: Ablauf Benachrichtigung empfangen

Anschliessend prüft, der NotificationService, ob Sprachsynthese lokal aktiviert ist. Ist diese deaktiviert, endet die Verarbeitung. Ist die Sprachsynthese lokal aktiviert, 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

[12] Vereinfachter Ablauf

überprüft, ob im Applikationsverzeichnis bereits eine MP3-Datei für den Empfangenen Benachrichtigungstyp (NotificationType) in der Version der Empfangenen 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. Der CloudService findet die Inhalte der relevanten Benachrichtigung in der Konfiguration und sendet eine Anfrage an den Sprachsynthese Provider. Anschliessend leitet er die Resultate des Providers an den Client weiter. Der Client speichert die empfangenen Daten lokal im Applikationsverzeichnis unter Id und Version des Benachrichtigungstyps (NotificationType). Nachdem die Daten gespeichert wurden, wird deren Inhalt abgespielt.

5.4 Gegensprechanlage

Mit dem Einbau von synchroner Sprachübertragung wird Praxisruf um die Funktion einer Gegensprechanalge 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.

5.4.1 Konfiguration

Die Gegensprechanalge wird in den nativen Mobile Client integriert. Praxismitarbeitende können über Buttons Sprachverbindungen zu anderen Clients aufbauen.[13] Welche Buttons und damit welche Sprachverbindungen zur Verfügung stehen, kann vom Administrator über das Admin UI konfiguriert werden. Damit dies möglich ist, sind Änderungen an der Configuration Domain des Cloudservice von Praxisruf sowie am Admin UI notwendig.

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. Analog dazu wird für den Aufbau von Sprachverbindungen eine 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.

Die Konfiguration eines Clients wird durch die bestehende Entität ClientConfiguration abgebildet. Diese Konfiguration wird um eine Liste von CallTypes erweitert. So kann für jeden Client definiert werden, welche Sprachverbindungen aufgebaut werden können.

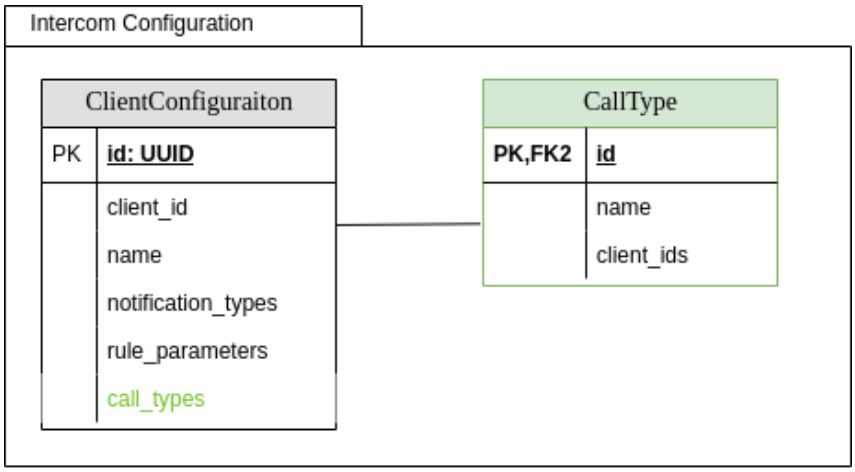


Abbildung 5.10: ERD Ausschnitt - Konfiguration Gegensprechanalge

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 entferntw erden können. Die bestehenden Endpunkte für ClientConfiguration werden entsprechend erweitert.

[13] Vlg. Kapitel 4.2.x

5.4.2 Signaling Server

Mit WebRTC werden Peer To Peer Sprachverbindungen aufgebaut.[14] Bevor die Verbindung aufgebaut ist, kennen die beiden Clients sich gegenseitig noch nicht. Die beteiligten Clients müssen Signale austauschen können um sich auf die Details der Verbindung zu einigen. Es braucht deshalb eine Instanz, welche beide Seiten kennt und Vermittlung von Signalmeldungen übernimmt. In Praxisruf übernimmt der Cloudservice die Funktion, Benachrichtigungen zu vermitteln.

Prinzipiell ist es möglich, den Signalaustausch in dieses Modul zu integrieren. Dieser Ansatz hat aber zwei grosse Probleme. Erstens ist die Grösse von Medlungen, welche über den gewählten Messaging Service (Firebase Cloud Messaging) ausgetauscht werden können beschränkt. Meldungen zum Aufbau von WebRTC Verbindungen können diese Grösse überschreiten.[15] Zweitens geschieht das Versenden von Benachrichtigungen über einen asynchronen Mechanismus. Die Signale für Sprachverbindungen sollen aber synchron ausgetauscht werden. Es soll unmittelbar beim Versuch des Verbindungsaufbau klar sein, ob die Verbindung etabliert werden kann oder nicht.

Mit diesem Projekt wird der Cloudservice deshalb um ein weiteres Modul ”Signaling” erweitert, welches den austausch von Signalen zwischen Clients ermöglicht. Gleich wie das Modul für Sprachsynthese wird er unabhängig von den anderen Domänenmodulen im Cloudservice implementiert. Alle Informationen, welche aus anderen Domänen benötigt werden, werden über Rest-Schnittstellen bezogen. Das Modul Signaling übernimmt drittens Aufgaben. Erstens muss es den Clients die Möglichkeit bieten, sich für Sprachverbindungen zu registrieren und diese Registrierungen verwalten. Zweitens muss es Signalmeldungen empfangen und an die relevanten Empfänger zustellen. Drittens muss es wann immer möglich Clients über nicht zustellbare Signale benachrichtigen. Diese drei Funktionen müssen unabhängig von der gewählten Technologie für den Signalaustausch unterstützt werden. Es wird ein entsprechendes Interface spezifiziert[16]. Dieses definiert die Methoden afterConnectionEstablished und afterConnectionClosed. Die beiden Methoden werden aufgerufen, wenn eine Verbindung geöffnet resp. geschlossen wurde. Geöffnete Verbindungen werden in Memory geführt. Dabei wird jede Verbindung durch eine UUID identifiziert. Diese UUID entspricht der technischen Identifikation des entsprechenden Mobile Clients und muss, wenn sich der Mobile Client für eine Verbindung registriert mitgesendet werden. Ein Mobile Client ist für den Signalaustausch verfügbar, wenn das Signaling Modul eine Verbindung mit seiner technischen Id kennt. Sobald eine Verbindung geschlossen wurde, wird sie aus der Liste der in Memory Verbindungen wieder gelöscht. Der entsprechende Mobile Client steht damit nicht mehr für den Signalaustausch zu Verfügung.

Für das Zustellen von Signalen wird die Methode sendMessage definiert. Jede Message beinhaltet eine Payload und die Identifikation des Empfängers. Der Signaling Service durchsucht die bekannten Verbindungen und findet die Verbindung für die Identifikation des Empfängers. Anschliessend sendet er das Signal über die Verbindung dieses Empfängers.

Für die Verwaltung von bestehenden Verbindungen wird die Komponente ConnectionRegistry implementiert. Diese muss Verbindungen anhand einer id registrieren und Verbindungen wieder entfernen können. Weiter müssen Verbindungen anhand ihrer id gefunden werden können. Die ConnectionRegistry wird als Wrapper um eine Java HashMap implementiert. Durch die Verwendung der Map über einen Wrapper bleibt die Implementation aber unabhängig davon und kann leicht ausgetauscht werden. [17]

[14] citation
[15] cite
[16] Siehe Listing 6
[17] Siehe Listing 7

TODO: Replace with class diagram

```java
/**
 * Contracts for clients to register an intercom connection in Praxisruf
 * Once a connection is established it can be used, to negotiate messages between
 * registered clients. This enables signaling server functionality for when
 * establishing Peer To Peer Connections between clients.
 *
 * @param <T> Type of the connection
 * @param <M> Type of messages that will be exchanged
 */
public interface ClientConnector<T, M> {

    /**
     * Receives a message and forwards it to all relevant registered connections.
     * M is expected to contain the key any relevant connection.
     */
    void sendMessage(M message);

    /**
     * Is called after a connection has been established. The established connection
     * is stored in a ConnectionRegistry with key: clientId and value: connection.
     */
    void afterConnectionEstablished(UUID clientId, T connection);

    /**
     * Is called after a connection has been closed. The closed connection is
     * removed from the ConnectionRegistry.
     */
    void afterConnectionClosed(T connection);
}
```

Listing 6: ClientConnector.java

```java
/**
 * Contracts for managing connections created by ClientConnector.
 * @param <T> Type of the connections
 */
public interface ConnectionRegistry<T> {

    /**
     * Registers the given connection in a key value store using clientId as key.
     * @return boolean - whether the registration is registered
     */
    boolean register(UUID clientId, T connection);

    /**
     * Removes the given connection from the key value store
     * @return boolean - whether the registration is unregistered
     */
    boolean unregister(T connection);
}
```

Listing 7: ClientConnector.java

Die Schnittestelle des Signaling Servers nach aussen wird mit Websockets umgesetzt. Wie die restlichen Module des Cloudservices wird auch der Singaling Server als Spring Boot Modul umgesetzt. Mit dem Projekt Spring-Boot-Starter-Websocket, können Websocketanbindungen in einer Spring Boot Applikation umgesetzt werden.[18] Es wird ein WebsocketHandler implementiert, welcher unter dem Pfad "server-url/signaling" erreichbar ist. Die Implementation der Websocketschnittstelle implementiert das oben beschriebene ClientConnector Interface mit der entsprechenden Funktion.

Etablierte Verbindungen müssen eindeutig einem Client zugeordnet werden können. Diese Client Identifikation wird als Query Parameter bei Verbindungsaufbau mitgegeben und im WebsocketHandler ausgelesen. Anschliessend wird die WebsocketSession zusammen mit der Client Identifikation in der ConnectionRegistry gespeichert. Wird die WebsocketSession geschlossen, wird sie durch den WebsocketHandler wieder von aus der ConnectionRegistry entfernt.

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 Meachnismus wie für Http Anfragen in den anderen Cloudservice Domänen verwendet werden. Über die SecurityConfig des Cloudservices wird die Authentification aller Http Requests überprüft. Diese Konfiguration sorgt dafür, das unauthentifizierte Http Requests eine entsprechende Response erhalten. Sie verhindert allerdings nicht, dass eine Websocketverbindung aufgebaut wurde, obwohl der Request nicht authentifiziert werden konnte. Um dies zu verhindern, muss der Http Request für den Aufbau einer Websocketverbindung notwendig sind abgefangen werden und geprüft werden, ob der Request authentifiziert wurde.**cite** Spring bietet dazu die Klasse HttpSessionHandshakeInterceptor. Mit der Methode beforeHandshake kann der Request vor dem Verbindungsaufbau abgefangen werden und die Authentisierung überprüft werden. Ist diese gültig, wird die Verbindung aufgebaut. Andernfalls wird der Verbindungsaufbau abgebrochen und eine Reponse mit HttpStatue "unauthorized" zurückgesendet.

[18] cite

5.4.3 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 aller Clientconfigurations die dem aktuellen Benutzer zur Verfügung stehen beim Cloudservice angefragt. Der Benutzer wählt die gewünschte Konfiguration aus und bestätigt.

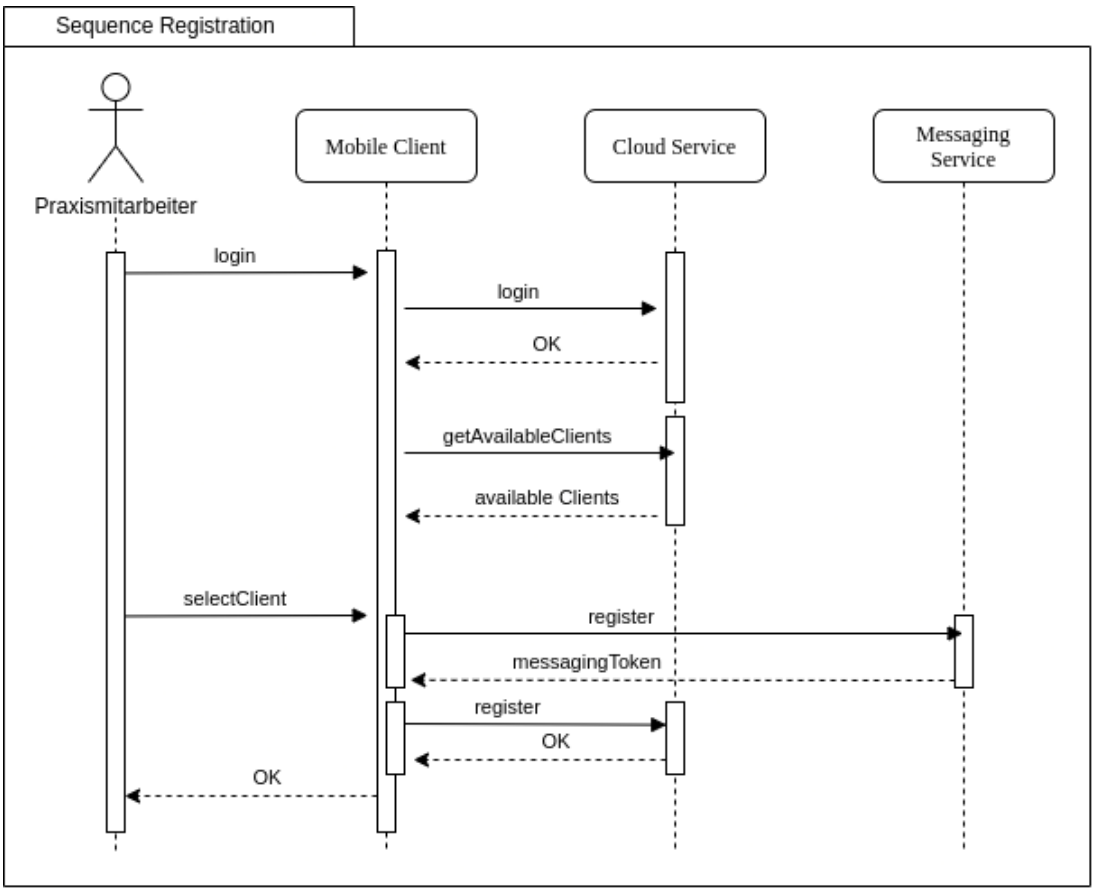


Abbildung 5.11: Mockup Home

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 beim Messaging Service. 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 bei sich und kann sie verwenden um Benachrichtigungen an diesen Client zuzustellen. Für Sprachverbindungen muss zudem eine Verbindung zum Signlaing Modul des Cloudservices aufgebaut werden. Dazu wird wie in Kapitel 5.4.5 beschrieben eine Websocketverbindung geöffnet.

5.4.4 Signalmeldungen

Dieses Kapitel beschreibt die Signalmeldungen, welche für Sprachverbindungen verwendet werden.

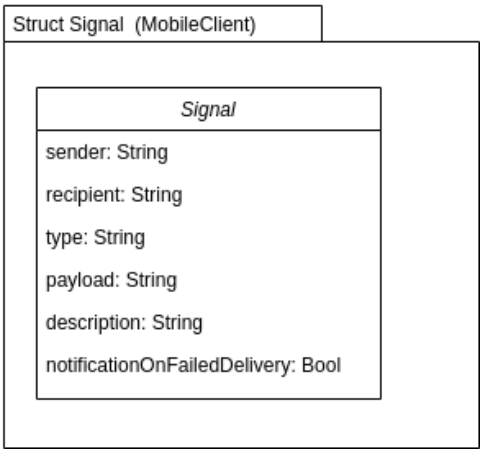


Abbildung 5.12: Signal (Mobile Client)

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.

OFFER: Ein Offer wird vom Initiator der Sprachverbindung an die Empfänger gesendet. Es beinhaltet die SDP[19] Informationen des Initiators.

ANSWER: Eine Answer wird vom Empfänger eines Offers an den Initiator der Sprachverbindung gesendet. Es beinhaltet SDP[20] Informationen des Empfängers.

ICE CANDIDATE: Ein RTCIceCandidate beschreibt Routing- und Protokollinformationen, die für den Verbindungsaufbau verwendet werden. Nach Verarbeitung von OFFER und ANSWER tauschen Initiator und Empfänger solange ICE CANDIATE Signale aus, bis sie sich auf einen Kandidaten geeinigt haben.

END: Wird von Empfänger oder Initiator versendet, nachdem die Verbindung durch tippen des Auflegen-Button in der Applikation beendet wurde. Empfang dieses Signal führt dazu, dass alle offene Sprachverbindungen beendet werden.

UNVAILABLE: Wenn der Signalinvserver ein Signal nicht zustellen kann, wird das Ziel wenn möglich über eine Benachrichtigung informiert. Zudem wird ein Unavailable Signal zurück an den Sender gesendet. So weiss dieser, dass der gewünschte gesprächspartner nicht verfügbar ist.

DECLINE: Praxismitarbeitende haben die Möglichkeit den Empfang von Anrufen zu deaktivieren. Wird ein Signal empfangen während Anrufe deaktiviert sind, sender der Mobile Client ein DECLINE Signal zurück.

[19] Sesseion Description Protocol
[20] Sesseion Description Protocol

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 mehrere Signalmeldungen ausgetauscht werden. Der auslösende Client initialisiert die Peer to Peer Verbindung auf seiner Seite und sendet für jeden Gesprächspartner ein Angebot (OFFER), welches signalisiert, dass er eine Sprachverbindung aufbauen möchte. Der Cloudservice findet die Verbindung des relevanten Empfängers und leitet das Signal über diese Verbindung weiter. Der Empfänger nimmt das Angebot (OFFER) entgegen, initialisiert die Verbindung auf seiner Seite und sendet eine Antwort (ANSWER). Diese wird wiederum über den Cloudservice weitergeleitet. Der auslösende Client empfängt die Antwort (ANSWER) und ergänzt die notwendigen Verbindungsinformationen.

Folgende Graphik zeigt den initialen Ablauf des Verbindungsaufbau:

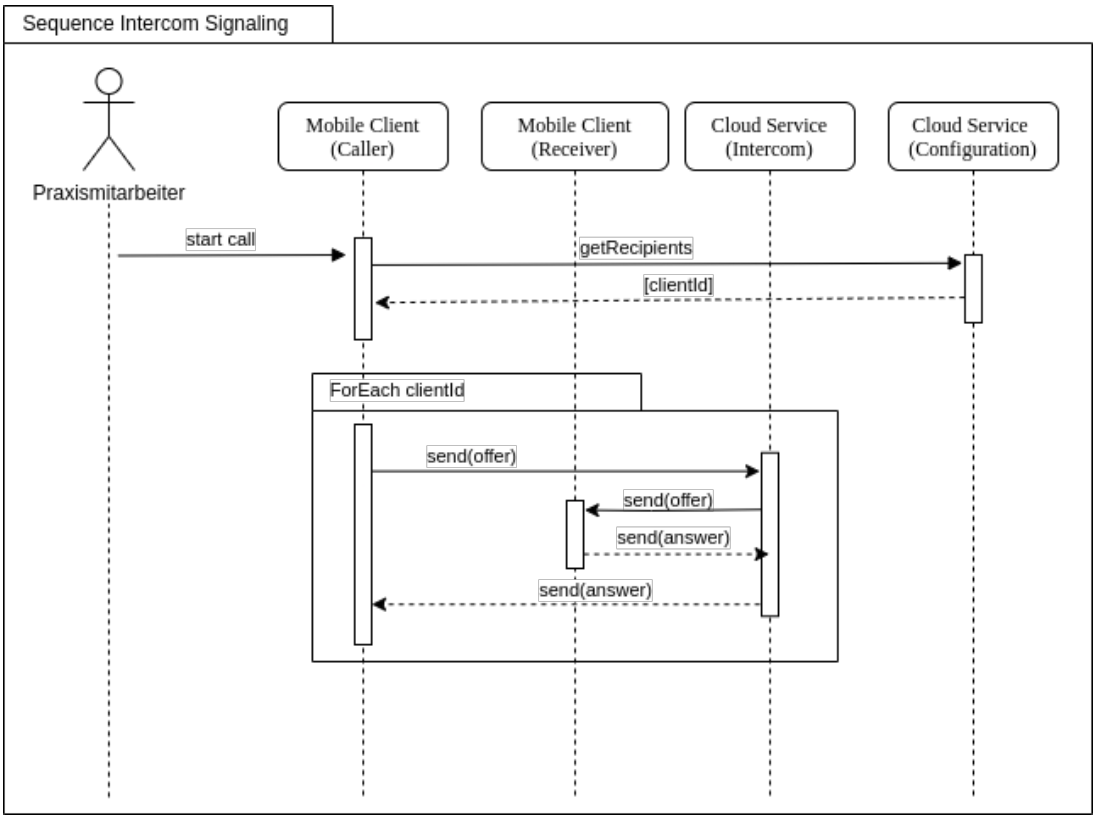


Abbildung 5.13: Ablauf Verbindungsaufbau Gegensprechanalge

Der Mobile Client kennt nun die technischen Identifikatoren der Clients, zu denen eine Verbindung aufgebaut werden soll. Um die Peer To Peer Verbindung zu diesen Clients aufzubauen, müssen nun weitere Nachrichten mit dem Cloudservice ausgetauscht werden. Alle verfügbaren Clients haben sich bei der Anmeldung mit dem Cloudservice verbunden. Der Cloudservice führt eine Liste über die verfügbaren Verbindungen und die dazugehörigen technischen Identifikatoren. Über diese Verbindungen werden nun die Nachrichten ausgetauscht die zum aufbauen der WebRTC Verbindung benötigt werden. Der Austausch dieser Nachrichten folgt dem Interactive Connection Establishment Protokoll (ICE). Der Client initialisiert für jede der erhaltenen client Ids einen Rtc Connection. Dies dient als Endpunkt der Connection auf seiner Seite. Anschliessend Sendet der Client ein Anfrage an den Cloudservice. Dieses Anfrage

beinhaltet das ICE Offer und die Client Ids des von Ausgangs- und Ziel-Client. Der Cloudservice findet die Verbindung des Zieles anhand der Client Id und leitet das ICE Offer und Ausgangs Client Id über die registrierte Verbindung an den Empfänger weiter. Wenn der Empfänger das ICE Offer erhält, initialisiert er auf seiner Seite die Rtc Connection und sendet eine Anfrage mit ICE Antwort und originaler Ausgangs Client Id an den Server zurück. Dieser leitet die Anfrage dann analog dem ICE Offer an die Verbindung des ausgehenden Clients weiter. Sobald dieser die Antwort erhält, ist die Rtc Connection bestätigt und die Sprachverbindung ist initialisiert.

5.4.5 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.[21] 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[22] verzichtet. Stattdessen wird eine Extension Klasse PraxisrufApi+Signaling nach folgendem Muster erstellt:

```
1  protocol PraxisrufApiSignalingDelegate {
2      func onConnectionLost()
3      func onSignalReceived(_ signal: Signal)
4      func onErrorReceived(error: Error)
5  }
6
7  extension PraxisrufApi {
8
9      static var signalingDelegate: PraxisrufApiSignalingDelegate?
10
11     private static var signalingWebSocket: URLSessionWebSocketTask? = nil;
12
13     private var disconnected: Bool {
14         ...
15     }
16
17     func connectSignalingServer(clientId: String) {
18        ...
19     }
20
21     func disconnectSignalingService() {
22         ...
23     }
24
25     func pingSignalingConnection() {
26         ...
27     }
28
29     func sendSignal(signal: Signal) {
30         ...
31     }
32
33     func listenForSignal() {
34         ...
35     }
36 }
```

Listing 8: PraxisrufApi+Signaling.swift

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 ”disconnected”intern abgefragt werden. Die Extension bietet Methoden um den Signaling Service zu Verbin-

[21] Vgl. Kapitel 5.4
[22] Vgl. Kapitel 5.2.2

den. Eine Verbindung wird dabei immer spezifisch für die aktuell gewählte Client Configuration geöffnet. [23] 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.

```
1 protocol PraxisrufApiSignalingDelegate {
2     func onConnectionLost()
3     func onSignalReceived(_ signal: Signal)
4     func onErrorReceived(error: Error)
5 }
```

Listing 9: PraxisrufApiSignalingDelegate.swift

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.

[23] Vgl. Kapitel X.X

5.4.6 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.

```
1 protocol CallClientDelegate {
2     func send(_ signal: Signal)
3     func updateState(clientId: String, state: String)
4     func onIncommingCallPending(signal: Signal)
5     func onIncomingCallDeclined(signal: Signal)
6     func onCallEnded()
7 }
```

Listing 10: CallClientDelegate.swift

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.

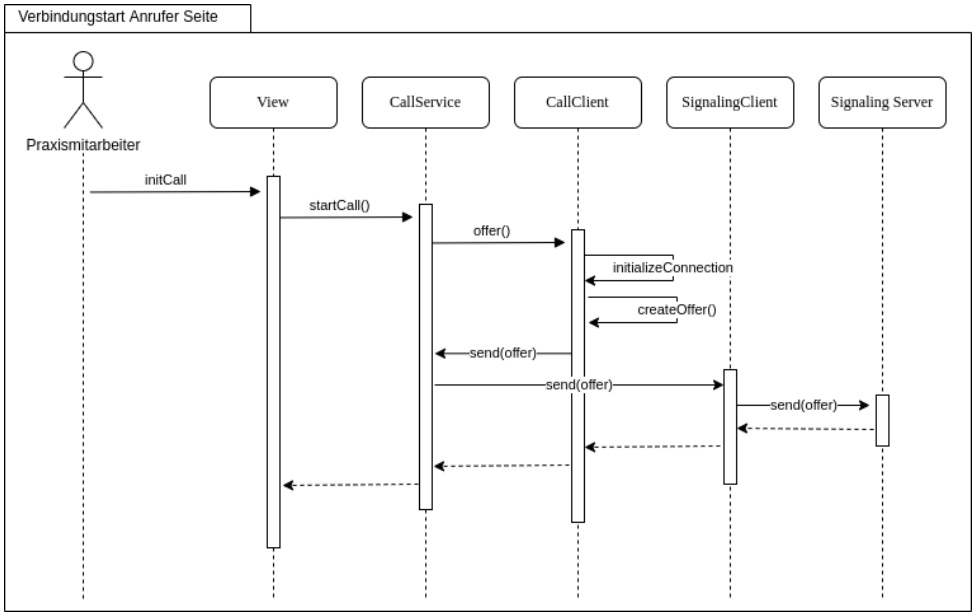


Abbildung 5.14: MobileClient - Anruf Starten Signal

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.

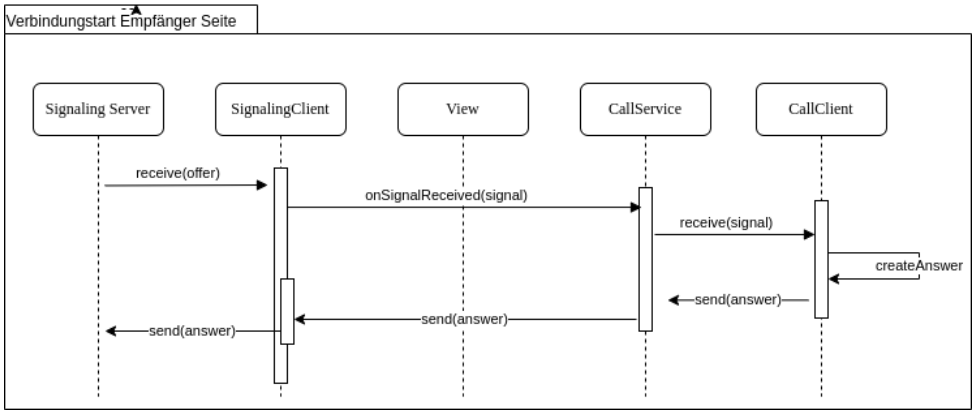


Abbildung 5.15: Mobile Client Anruf Empfangen Signal

5.4.7 Integration WebRTC in Mobile Client

Gekapselt im Service CallClient. Managed den State von Sprachverbindungen. Kommuniziert nur über den Delegate mit dem Rest der Applikation. Verwendet die Klassen der WebRTC iOS Library um Peer To Peer Verbindungen zu machen. Die dazu notwendigen Signale werden über Delegate / Signaling Server ausgetauscht.

5.4.8 Verpasste und vergangene Anrufe

Verpasste geben Push Benachrichtigung Verpasste müssen quittiert werden. Vergangene erscheinen in Inbox. Vergangene müssen nicht quittiert werden.

5.4.9 Unterhaltung

Mute Button Audio Off Button End Call Button

5.4.10 Verbindungsende

Button antippen

5.5 Zusammenfassung

Domänenmodell Cloudservice

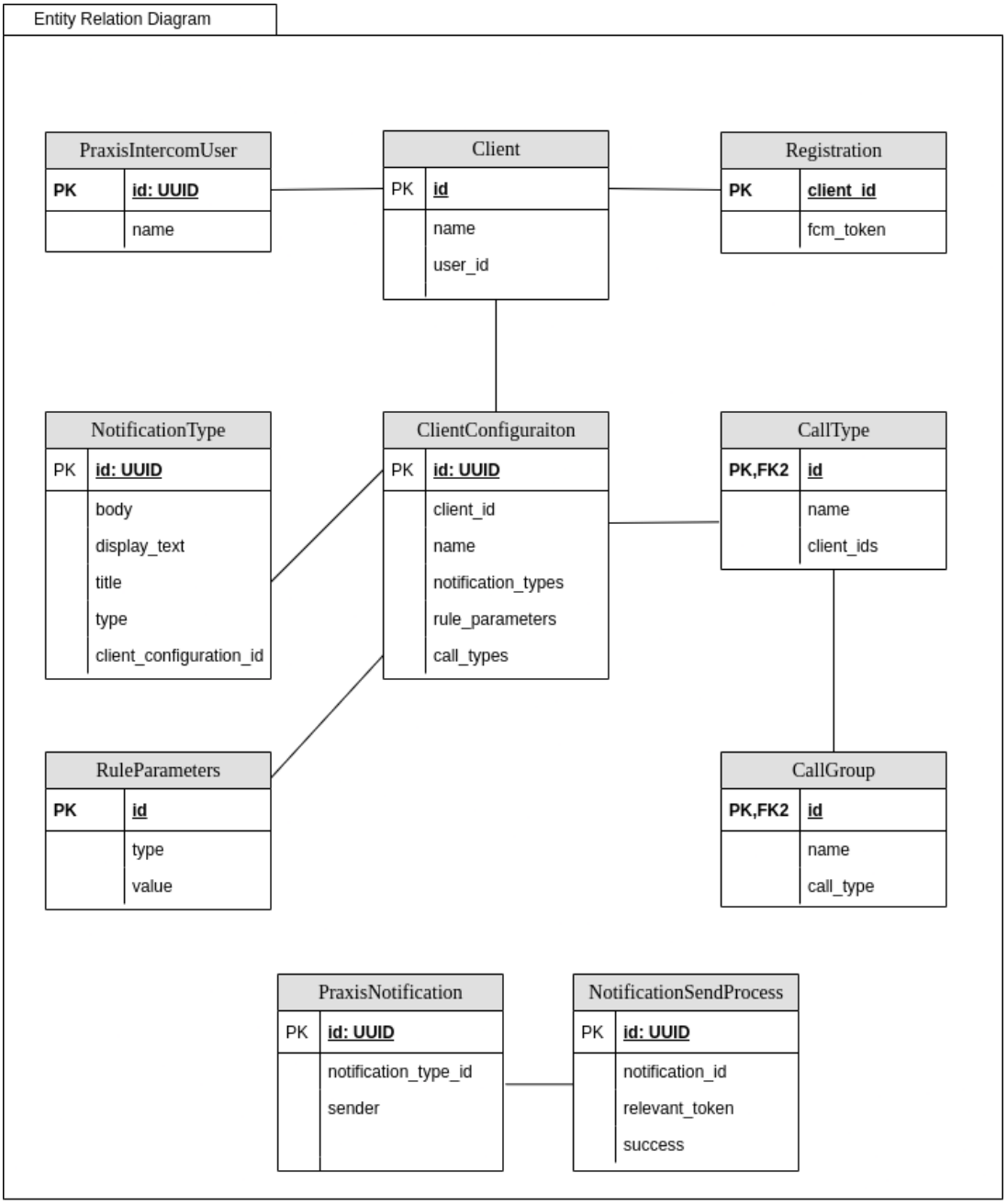


Abbildung 5.16: Entitiy Relation Diagramm - Cloudservice

API Cloudservice

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.

6 Umsetzung

6.1 Resultate

Lorem ipsum

6.2 Tests

6.2.1 Benutzertests

Lorem ipsum

6.2.2 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	+
S02	Benachrichtigung versenden - kein Empfänger	+
S03	Benachrichtigung empfangen.	+
S04	Fehler beim Versenden anzeigen.	+
S05	Wiederholen im Fehlerfall bestätigen.	+
S06	Wiederholen im Fehlerfall abbrechen.	+
S07	Audiosignal bei Benachrichtigung.	+
S08	Push Benachrichtigung im Hintergrund.	+
S09	Erinnerungston für nicht Quittierte Benachrichtigungen.	+
S10	Start Mobile Client - nicht angemeldet	+
S11	Start Mobile Client - angemeldet	+
S12	Anmelden mit korrekten Daten.	+
S13	Anmeldung mit ungültigen Daten.	+
S14	Konfiguration Wählen	+
S15	Abmelden.	+
S16	Admin UI - Anmeldung mit korrekten Daten	+
S17	Admin UI - Anmeldung mit ungültigen Daten	+
S18	Admin UI - Konfiguration Verwalten	+

6.2.3 Testplan Sprachsynthese

6.2.4 Testplan Gegensprechanlage

6.2.5 Testplan Performance

6.3 Fazit

Lorem ipsum

7 Schluss

Lorem Ipsum

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] OpenJS Foundation. (4. Jan. 2022). How NativeScript Works, Adresse: https://v7.docs.nativescript.org/core-concepts/technical-overview.

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

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

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

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

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

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

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

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

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

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

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

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

[16] ——, (). Amazon Polly Java Example, Adresse: https://docs.aws.amazon.com/polly/latest/dg/examples-java.html.

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

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

[19] Google Developers. (). WebRTC - Echtzeitkommunikation für das Web, Adresse: https://webrtc.org/.

[20] ——, (). WebRTC Mesh, Adresse: https://webrtcglossary.com/mesh/.

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

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

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

Abbildungsverzeichnis

A Aufgabenstellung

Fachhochschule Nordwestschweiz

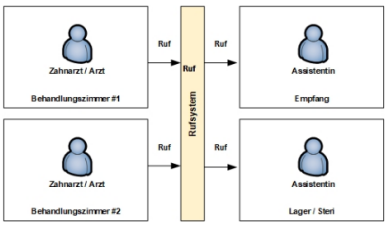
App
Web

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[24]. Berechtigungen können bei Joshua Villing angefordert werden.

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

C 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 .....