Fachhochschule Nordwestschweiz
Hochschule für Technik

Peer-to-Peer Kommunikation für Sprachübertragung in einem Praxisrufsystem

IP6 - Bachelor Thesis

23. November 2021

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]

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

2 Vorgehensweise

Projektplan

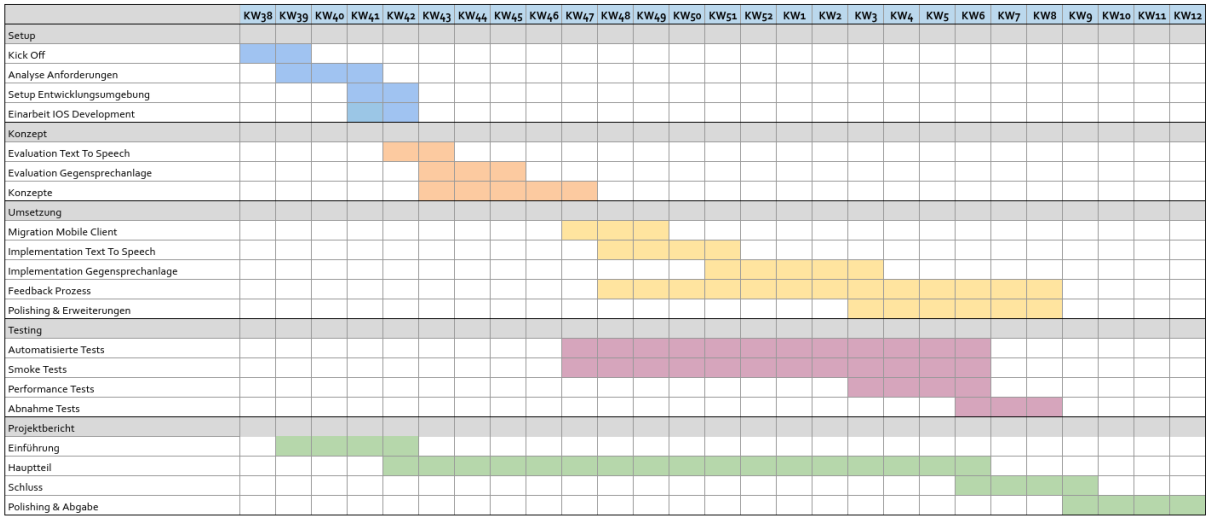


Abbildung 2.1: Projektplan

Lorem ipsum

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 Text To Speech 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 Text To Speech** Alle Anforderungen zu der Text To Speech 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	**Abnahme** Die Abnahmetests wurden zusammen mit dem Kunden ausgeführt.

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.

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	Text To Speech
F03	Gegensprechanlage

4 Technologie Evaluation

In diesem Kapitel wird evaluiert, mit welchen Technologien und Frameworks die Anforderungen für das Projekt umgesetzt werden. Dies beinhaltet die migration der bestehenden Mobile App[2] sowie die Implementation der neuen Features Text To Speechünd ”Gegensprechanlage”.

4.1 Mobile Client

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. Die bestehende Applikation wurde mit Nativescript 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, Hintergrundtasks zu erstellen und Audiosignale abzuspielen. Dadurch wird es möglich, regelmässig ein Signal abzuspielen um die Praxismitarbeitenden an verpasste Benachrichtigungen zu erinnern.

Programmiersprache

Für die Entwicklung von nativen iOS Applikationen ist die Programmiersprache Swift als Standard gesetzt.[3]

Frameworks

Für die Umsetzung von iOS Applikationen stellt Apple die zwei Frameworks UIKit[4] und SwiftUI[5] 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. Es hat allerdings den Nachteil das es schwerer zu erlernen und langsamer zu schreiben ist. (Citation Needed)

SwiftUI ist deutlich neuer als UIKit und steht seit iOS 13.0 zur Verfügung. Es hat eine tiefere Einstiegshürde als UIKit und ist grundsätzlich einfacher zu schreiben und warten (Citation Needed). In den Worten von Apple selbst: SSwiftUI helps you build great-looking apps across all Apple platforms with the power of Swift — and as little code as possible.”[5]

SwiftUI bietet zudem ausgezeichnete Integration des Entwicklungsworkflows in die XCode Entwicklungsumgebung. Es bietet schnelle live previews der Komponenten die geschrieben werden. Dies vereinfacht Design und Umsetzung der Ansichten.

Unterstützung Features

Die Ansichten aus dem bestehenden Mobile Client können mit SwiftUI und UIKit umgesetzt werden. SwiftUI bietet weiter viele Standardkomponenten wie Listenansichten, Formfelder und andere UIKomponenten, die es einfacher machen eine Benutzeroberfläche zu erstellen die den Look und Feel einer nativen iOS Applikation hat.

Die funktionalen Anforderungen zum Versenden und Empfangen von Benachrichtigungen sowie dem Abspielen von regelmässigen Erinnerungstönen können unabhängig vom gewählten UI Framework umgesetzt werden.

Für die Integration von Firebase Cloud Messaging stellt Firebase eine iOS Library zur Verfügung.[6] Die Registrierung bei Messaging Service[2] sowie das Versenden der Benachrichtigung kann direkt in über Services, welche die Library zur Verfügung stellt gemacht werden. Das Empfangen von Benachrichtigungen, benötigt Integration mit dem iOS Betriebssystem (citation needed). Mit sogenannten AppDelegates[7] ist es möglich sich in den Lifecycle des Betriebssystems einzuhängen und auf entsprechende Events zu hören. Um Hintergrundbenachrichtigungen empfangen zu können, muss die Methode "HERE$_G OES_T HE_M ETHOD$"$von UIApplicationDelegateverwendetwerden.AppDelegatessindeinKonzeptw

Erinnerungen können mit Boardmitteln aus den Libraries die Apple zur Verfügung stellt umgesetzt werden. Einerseits können mit Timer[8] regelmässig wiederholbare Tasks erfasst werden, die ausgeführt werden, wenn die App geladen ist. Weiter können über die Klasse BGTaskScheduler[9] Tasks erfasst werden, die im Hintergrund ausgeführt werden.

Entscheid

Der native iOS Mobile Client für Praxisruf wird mit Swift und SwiftUI umgesetzt. Als Zielplatform wird IOS15 verwendet, damit möglichst alle Funktionen aus SwiftUI zur Verfügung stehen. SwiftUI ist der neue Standard oder zumindest die Richtung in die Apple pushed. SwiftUI ist leichtgewichtiger und flexibler als UIKit. Es bietet ausgezeichnete Integration mit der Entwicklungsumgebung XCode und Entwicklungstools zur Umsetzung von Benutzeroberflächen mit nativem Look und Feel. Es ist davon auszugehen, dass die Entwicklung dadurch schneller und der resultierende Code schlanker und wartbarer ist.

SwiftUI ist neuer als UIKit und hat dementsprechend noch nicht denselben Umfang. SwiftUI ist aber mit UIKit kompatibel. Das heisst für Funktionen, die nur mit UIKIt umgesetzt werden können, kann UIKit verwendet werden. Diese Teile der Applikation können in Zukunft, wenn diese Funktionen mit SwiftUI umgesetzt werden migriert werden.

[2] Vgl. IP5 Kapitel X

4.2 Sprachsynthese

Praxisruf soll es neu Unterstützen, dass empfangene Benachrichtigungen vorgelesen werden. Um dies zu ermöglichen muss eine Technologie integriert werden, die es erlaubt den Textinhalt einer Benachrichtigung in Audio zu konvertieren welches abgespielt werden kann.

Grundsätzlich gibt es zwei Varianten wie dies erreicht werden kann. Die erste Option ist es, die Konvertierung auf dem Mobile Client selbst vorzunehmen. In diesem Fall ist die Konvertierung teil der Mobile App. Das IOS Core Framework bietet Packages welches dies erlauben.

Die zweite Option ist es, die Konvertierung an einen Cloud Service zu delegieren. Dazu muss ein externer Text To Speech Provider angebunden werden, der vom Praxisruf angesprochen werden kann.

Nativ in Mobile Client

Die Standardbibliothek von für iOS unterstützt das Konvertieren von Text zu Sprache.[10] Dementsprechend ist es mit Boardmitteln von iOS möglich, Sprach Synthetisierung umzusetzten. Diese Variante bietet den Vorteil, dass die Synthese ohne das Einbinden von weiteren Frameworks umgesetzt werden kann. Es ist weiter garantiert, dass die Funktion mit iOS funktionieren wird. Da die Funktion von Apple selbst zur Verfügung gestellt wird.

Diese Variante hat allerdings den Nachteil, dass eine starke Bindung zu Apple stattfindet. Sollte Apple sich je entscheiden, diese Funktion nicht mehr zur Verfügung zu stellen, muss die ganze Funktionalität auf einen anderen Provider umgeschrieben werden. Weiter hat die Variante den Nachteil, dass sie mehr Funktionalität in den Mobile Client auslagert. Die Verantwortung des Mobile Clients wird weniger klar getrennt. Er wäre nicht nur für die Interaktion mit dem Benutzer verantwortlich sonder muss die fachliche Anforderung der Sprachsynthese übernehmen. Letztlich hat diese Variante den Nachteil, dass dieselbe Funktionaltät für einen Android Client komplett neu entwickelt werden müsste.

Externer Provider in Mobile Client

Als zweite Option ist es möglich, die Sprachsynthese an einen externen Provider zu delegieren. Mit AWS Polly[11] ist es möglich, den Provider direkt aus einer iOS Applikation anzusprechen. Diese Variante hat den Vorteil, dass keine Bindung mehr zu Apple vorhanden ist. Für die Integration mit Android besteht hier ein wenig mehr Synergie. Es kann für iOS derselbe Provider verwendet werden. Da AWS Polly auch für Android einen SDK bietet. Diese Variante hat aber trotzdem noch die Nachteile, dass die Verantwortung des Mobile Clients grösser wird und dass unterschiedliche SDKs für die Android und iOS Plattformen verwendet werden.

Externer Provider über Cloud Service

Als Dritte Variante ist es möglich, die Sprachsynthese im Cloud Service vorzunehmen. Da AWS Polly auch einen SDK für Java bietet, ist es möglich, die Synthese dort einzubinden. 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 z.B. möglich einen externen File Storage anzubinden auf dem Audiodaten gespeichert werden. Dadurch muss der Sprachsynthese Provider weniger oft angesprochen werden. Die Variante hat den Nachteil, dass der Cloud Service komplexer wird. Durch das Hinzufügen von neuer Funktionaltiät ist das auf Systemebene aber so oder so gegeben. Der Einfluss davon kan weiter minimiert werden, indem

die neue Funktionalität gekapselt wird. Sie kann so umgesetzt werden, dass sie unabhängig von restlichen system ist und alle nötigen Daten über die Schnittstelle der anderen Module bezieht. [3]

Entscheidung

Die Sprachsynthese wird durch die Anbindung des externen Providers AWS Polly umgesetzt. Der Cloud Service übernimmt die Kommunikation mit AWS Polly und bietet eine Schnittstelle über die der Mobile Client Audiodaten beziehen knan.

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

[3] Vgl. Kapitel Systemarchitektur

4.3 Sprachübertragung

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

4.3.1 WebRTC

Mit WebRTC können Sie Ihrer Anwendung Echtzeit-Kommunikationsfunktionen hinzufügen, die auf einem offenen Standard basieren. Es unterstützt Video-, Sprach- und generische Daten, die zwischen Peers gesendet werden, sodass Entwickler leistungsstarke Sprach- und Videokommunikationslösungen erstellen können. Die Technologie ist in allen modernen Browsern sowie auf nativen Clients für alle wichtigen Plattformen verfügbar. 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."[12]

Varianten
Externer Anbieter (twilio)

Vorteile:

Einfachere Integration auf Client Seite durch Vendor SDK
Auf Cloud Service Seite nur Configuration Domain betroffen, keine andere Erweiterung nötig
Alles andere kann auf Client Seite erledigt werden.

Nachteile:
Kompliziertere Integration mit Client -¿ Client muss korrekte Verbindungen anhand Button Konfiguration aufmachen
Verantwortung des Clients wächst. Aktuell hat er nur Verantwortung zum Empfangen, die Arbeit wird immer von andern gemacht

Self Hosted (WebRtc Server als Teil von cloud service implementieren)

Vorteile:

Vermittlung und Signaling kann vom Cloud Service übernommen werden.
Grössere Flexibilität, da Vermittlung in Cloud Serivce
Evtl. Synergie mit Rules Engine in Configuration Domain

Nachteile:

Kompliziertere Einbindung auf Client Seite.

4.3.2 Meeting Solution

AWS Chime, Teams oder Ähnliches als Basis
Nicht wirklich p2p?
Starke Vendor Bindung

Full on VOIP mit Telefonnummern

Not feasible
Nicht was wir brauchen, da nur vordefinierte calls zwischen clients gemäss konfiguration möglich sein sollen
Es ist nicht gefordert und nicht gewünscht dass ïrgendjemandper telefonnummer, den client erreichen kann.

4.3.3 Entscheidung

WebRTC self hosted.

Synergie Rules Engine
Dispatching in Verantwortung Cloud Service
SDK gut genug
Nicht vendor abhängig

5 Konzept

5.1 SystemArchitektur

Mit diesem Projekt wird Praxisruf um die Funktionen Text To Speech und Gegensprechanalge erweitert. Um dies zu ermöglichen, sind Erweiterungen an der Systemarchitektur nötig. Bestehende Module bleiben. Modularität wird erweitert. Bisher nur Package Trennung. Neu werden Gradle Module pro Domain gemacht. Immernoch in einem Service nachher. Aber eine Stufe näher daran, es in Microservices aufzutrennen. Übersichtlicher, einfacher erweitertbar. Trennung der Domänen garantiert.

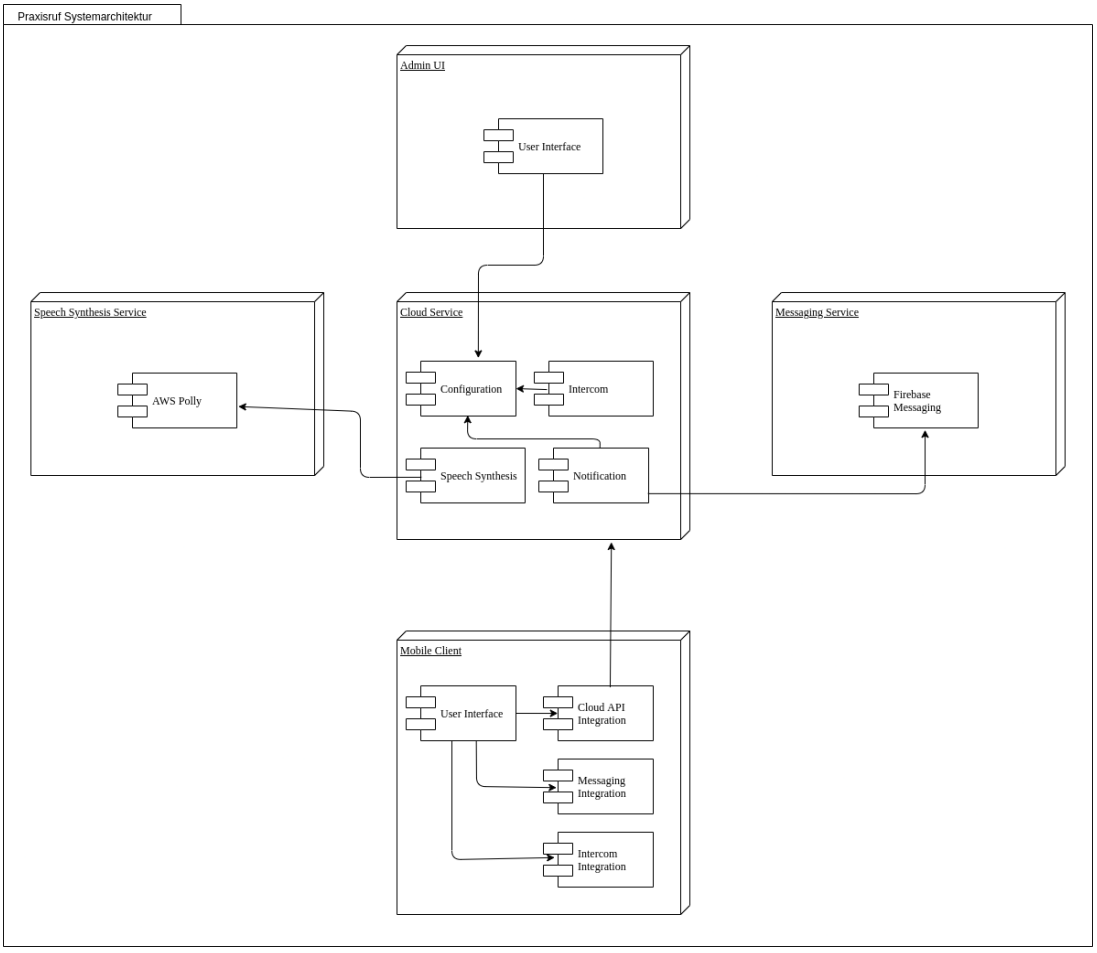


Abbildung 5.1: Systemarchitektur Praxisruf

Cloudservice

Auf Seite Cloud Service werden die Module Intercom und Speech Synthesis hinzugefügt. Intercom übernimmt das Signaling für WebRTC. Hat den Vorteil, dass künftig auch Web und Android Clients an denselben Signaling Service angebunden werden können. Vermittlung passiert anhand der vom Admin erfassten Konfiguration.

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

Neu als nativer Client mit SwiftUI. Beinhaltet des Benutzer Interface. Sowie Komponenten zur Anbindung an Configuration, Notification, Speech Synthesis und Intercom. Details in Client Kapitel.

Admin UI

Das Admin UI dient weiterhin zur Administaration der Configuration. Das Admin UI wird um Konfigurationsmöglichkeiten für Speech Synthesis und Gegensprechanalge werweitert.

Speech Synthesis Service

Als neuer externer Service AWS Polly angebunden. Dabei handelt es sich um die Speech Synthesis Funktion von Amazon Webservices.

Messaging Service

Der Messaging Service wird weiterhin zum Versenden von Benachrichtigungen verwendet. Am Messaging Service werden in diesem Projekt keine Änderungen vorgenommen.

5.2 Migration Benachrichtigungen

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.

Benutzeroberfläche

SwiftUI bietet alles was man braucht.

Anbindung Cloud Service

Anbindung an REST Schnittstellen ist mit SwiftUI natürlich Möglich. Ein zentraler API Service wird erstellt. Pro Domain die angesprochen wird, wird eine Extension erstellt. Der Api Service macht den Rest call, setzt authentication. Es muss für jeden Api Call ein Callback mitgegeben werden, dass bei completion ausgeführt wird. Dabei muss dieses Callback den Erfolgs und den Fehlerfall behandeln.

Integration in die Benutzeroberfläche funktionert über einen zwischengeschalteten Service (ViewModel?). Dieses verwendet @ObservableObject um die View über den SwiftUI Lifecycle zu aktualisieren.

Sämtliche Calls und Abläufe können mit diesen Mitteln analog zu IP5 umgesetzt werden.

Anbindung Firebase

Die Anbindung von Firebase ist nicht rein mit SwiftUI möglich. Wir benötigen die Lifecycle Integration zu iOS, die mit den AppDelegates von UIKit möglich ist. Auch mit SwiftUI können AppDelegates verwendet werden. Die Anbindung an Firebase Messaging wird dementsprechend mit AppDelegates gelöst. Die Anbindung erfolgt damit wie in der offiziellen Dokumentation vorgesehen. Damit die Abhängigkeit zu AppDelegates minimiert ist, sollen innerhalb der AppDelegates nur minimale Logik ausgeführt werden. Die echte Logik wird an unabhängige Services delegiert.

Scheduled Reminder für Inbox

IOS Development unterstützt scheduled tasks.

5.3 Sprachsynthese

Benutzeroberfläche

Erweiterung Inbox um T2S Icon.
Erweiterung Admin UI um Checkbox.
Zusätzlich Configuration Page mit preferences.

Konfiguration

Erweiterung NotificationType um ein boolean Flag für isTextToSpeech. Wenn Aktiviert, wird Benachrichtigung bei Empfang vorgelesen.

Weiter wird auf dem NotificationType ein Feld version hinzugefügt. Damit soll die aktuelle Version des NotificationType verfolgt werden. Wird eine Änderung am NotificationType persisteirt, wird die Version entsprechend angepasst. Sowohl das Flag isTextToSpeech als auch das version Property werden neu beim Versenden von Benachrichtigungen mitgegeben.

Text To Speech kann auf Client Seite dekativiert werden. Ist es deaktiviert, werden keine Benachrichtigungen vorgelesen. Audio Signal, dass Benachrichtung empfangen wurde ertönt aber trotzdem. Keine zusätzlichen Endpoints am Cloud Service nötig.

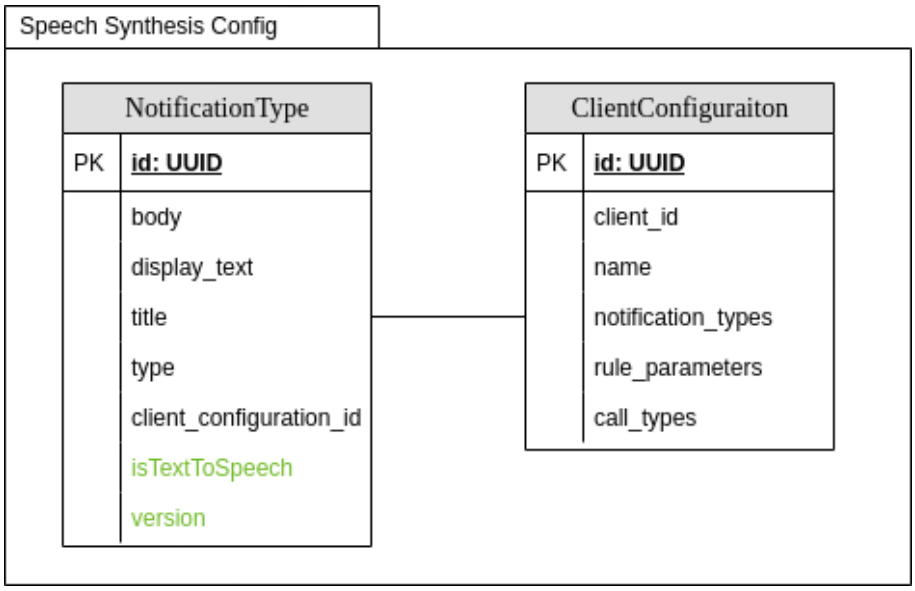


Abbildung 5.2: ERD Ausschnitt - Konfiguration Sprachsynthese

Anbindung Sprachsynthese Service

Die Anbindung des Sprachsynthese Service erfolg zentral über den Cloud Service. Dazu wird der Cloud Service um ein modul Speech erweitert. Dieses Modul stellt einen Endpoint zur verfügung über den die Sprachdaten zu einer Benachrichtigung abgefragt werden können. Einziger Parameter für diese Anfrage ist die technische Identifikation des Benachrichtigungstyps der relevanten Benachrichtigung. Das Speech-Modul kann anhand dieser Identifikation den NotificationType beim Configuration-Modul abholen. Dem NotificationType kann der Text entonmmen werden, der snytetisiert werden soll. Dieser Text kann dann an den Sprachsynthese Service gesendet werden. Das Resultat dieser Anfrage wird als Resultat der Abfrage vom Client zurückgegeben.

```
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 1: SpeechSynthesisController.java

Auf Client seite kann dieser Endpunkt nun als Download angesprochen werden.

```
1  extension PraxisrufApi {
2      func synthesize(authToken: String, completion: @escaping (Result<URL,
       PraxisrufApiError>) -> Void) {
3          guard let audioUrl = URL(string: "\(baseUrlValue)/speechsynthesis") else {
4              completion(.failure(.custom(errorMessage: "Invalid url configuration")))
5              return
6          }
7          var request = URLRequest(url: audioUrl)
8          request.addValue("Bearer \(authToken)", forHTTPHeaderField: "Authorization")
9          URLSession.shared.downloadTask(with: request) { result, response, error in
10             guard let audioFileLocation = result else {
11                 completion(.failure(.custom(errorMessage: "No audio received")))
12                 return
13             }
14             completion(.success(audioFileLocation))
15         }.resume()
16     }
17 }
```

Listing 2: PraxisrufApi+Speech.swift

Um die Anbindung im Cloud Service vom konkreten Anbieter möglichst unabhängig zu machen, wird die Integration an den Speech Synthese Service über ein Interface abstrahiert. Soll ein neuer Sprachsyntese Service angebunden werden, muss lediglich dieses Interface für den neuen Service implementiert werden und alles andere kann unberührt bleiben.

```java
/**
 * Contracts for getting speech synthesis data from a speech synthesis provider.
 */
public interface SpeechSynthesisService {

    /**
     * Requests the NotificationType with the given id from the configuration
     * and then sends a request to a speech synthesis provider.
     *
     * The result from the provider is written into a InputStreamResource
     * containing MP3 audio data.
     */
    InputStreamResource synthesize(UUID id);

}
```

Listing 3: SpeechSynthesisService.java

Für dieses Projekt wird AWS Polly verwendet. Dementsprechend wird das SprachSyntheseService Interface für AWS Polly implementiert. AWS Polly bietet einen Java SDK, welcher die Anbindung ermöglicht. Dieser SDK bietet alle Klassen die für die Anbindung an AWS Polly nötig sind. Da im Cloud Service Spring Boot verwendet wird, können die für die Verbindung nötigen Klassen mit einer Spring Config erstellt werden und dann über Dependency Injection im AwsPollySpeechSynthesis Service verwendet werden.

```java
@Configuration
@ConfigurationProperties(prefix = "praxis-intercom.aws")
public class AwsConfiguration {

    private String accessKey, secretKey, region, language;

    @Bean
    public AmazonPollyClient amazonPollyClient() {
        var credentialsProvider = credentialsProvider();
        var clientConfiguration = new ClientConfiguration();
        var polly = new AmazonPollyClient(credentialsProvider, clientConfiguration);
        polly.setRegion(region);
        return polly;
    }

    @Bean
    public Voice voice(AmazonPollyClient polly) {
        var describeVoicesRequest = new DescribeVoicesRequest();
        var describeVoicesResult = polly.describeVoices(describeVoicesRequest);
        return describeVoicesResult.getVoices().stream()
                .filter(v -> v.getLanguageName().equals(language)).findFirst().get();
    }

    private AWSStaticCredentialsProvider credentialsProvider() {
        var basicAWSCredentials = new BasicAWSCredentials(accessKey, secretKey);
        return new AWSStaticCredentialsProvider(basicAWSCredentials);
    }
}
```

Listing 4: AwsConfiguration.java

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

Laufzeitsicht

Empfang und Versenden gleich wie bei IP5. Benachrichtigung enthält zudem neu Flag ob T2S gebraucht werden soll. Wenn ja, wird Vorlesen an T2S Service delegiert.

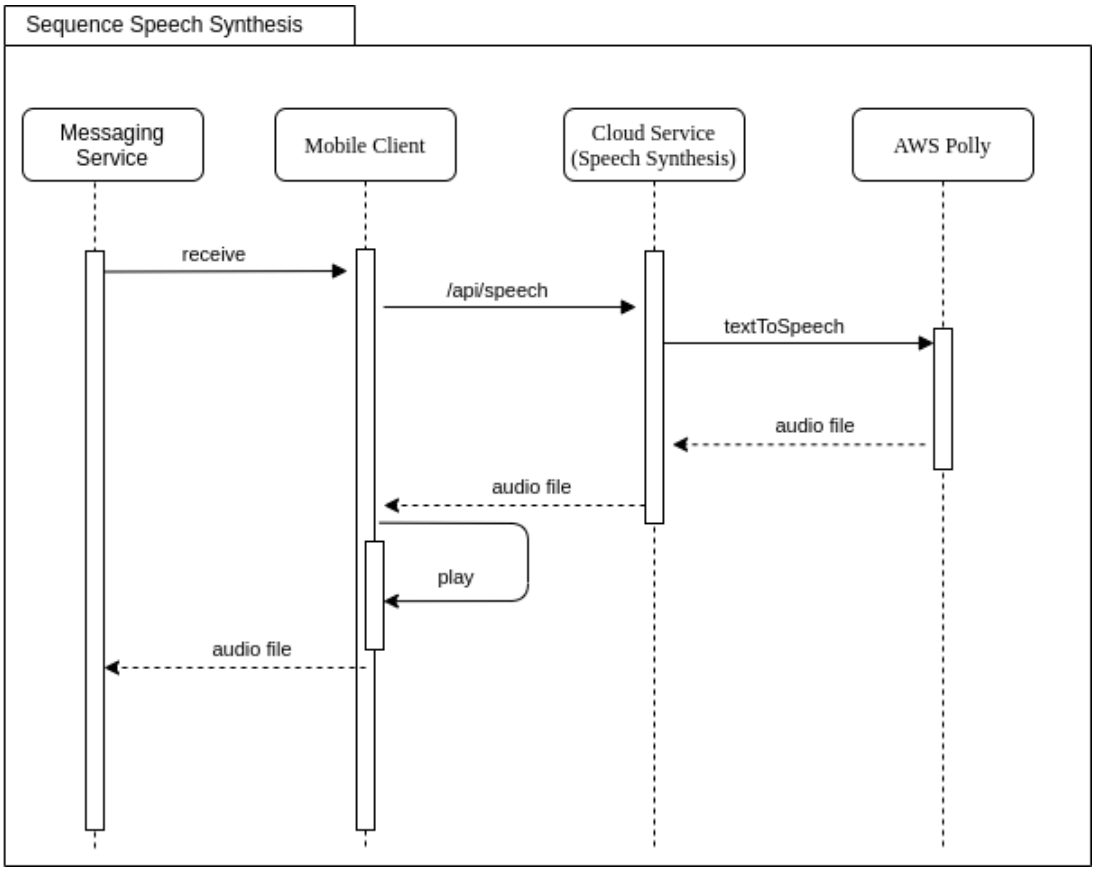


Abbildung 5.3: Ablauf Benachrichtigung empfangen

5.4 Sprachübertragung

Übersicht

Mit dem Einbau von synchroner Sprachübertragung wird Praxisruf um die Funktion einer Gegensprechanalge erweitert. Um Sprachverbindungen zu anderen Zimmern aufbauen zu können, wird die Ansicht der Mobilen Applikation um einen Bereich für die Gegensprechanalge erweitert. In diesem Bereich sind analog zum Bereich um Benachrichtigungen zu versenden Buttons vorhanden, über welche eine Sprachverbindung aufgebaut werden kann. Welche Buttons in einem Client zu Verfügung stehen muss vom Praxismitarbeitenden über das Admin UI konfiguriert werden. Die Konfiguration eines Buttons beinhaltet den Text der auf Clientseite angezeigt wird, sowie eine Liste zu welchen Clients Sprachverbindungen aufgebaut werden sollen.

Mit der gewählten Technologie WebRTC werden die Sprachverbindungen zwischen Clients Peer to Peer aufgebaut. Die Verbindungen werden dabei vom Client selbst initialisiert. Damit dies möglich ist, benötigt es einen Signaling Server, welcher die einzelnen verfügbaren Clients kennt und den Austausch der Daten die zum Verbindungsaufbau notwendig sind koordiniert.

Benutzeroberfläche

5.4.1 User Interface

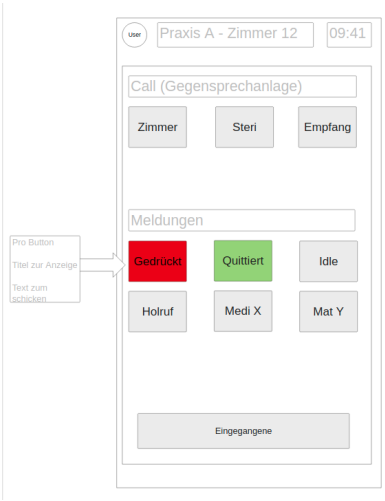


Abbildung 5.4: Mockup Home

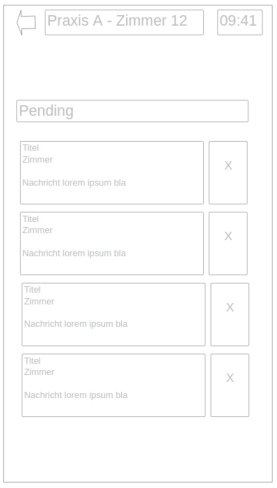


Abbildung 5.5: Mockup Inbox

Active Call Screen
Erweiterung Admin UI für Konfiguration CallType

Konfiguration

Praxisruf wird um die Funktion einer Gegensprechanlage erweitert. Dabei soll von einem Administrator zentral konfiguriert werden können, zwischen welchen Clients Sprachverbindungen aufgebaut werden können. Damit dies möglich ist, sind Änderungen an der Configuration Domain des Cloud Service von Praxisruf 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. Diese kann mehreren ClientConfigurations zugeordnet werden. 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. Es ist möglich, dass dieselbe Gruppe an Zielen auf verschiedenen Clients verschieden heissen soll. Um dies einfach zu ermöglichen, werden die Ziele der Unterhaltung in eine eigene Entität CallGroup ausgelagert. So kann pro Client ein CallType definiert werden der einen eigenen Anzeigtext definiert. Die CallGroup muss aber nur einmal erstellt werden und kann auf mehreren CallTypes verwendet werden. Die ClientConfiguration die bei der Anmeldung vom Mobile Client geladen wird, wird um diese CallTypes erweitert. Dabei werden für jeden CallType aber nur der technische Identifikator und der Anzeigename mitgegeben. Zu welchen Zielen Verbindungen aufgebaut werden sollen, wird bei jedem Verbindungsabbau erneut beim CloudService angefragt. Dies ermögicht es, dass Änderungen an der Konfiguration sofort angewendet werden, ohne dass die ganze Konfiguration neu geladen werden muss. Für die beiden neuen Entitäten werden Endpoints für CRUD Operationen analog zu den anderen Konfigurationsentitäten hinzugefügt. Zudem wird das Admin UI um Ansichten erweitert, um diese Entitäten anzuzeigen, bearbeiten und löschen.

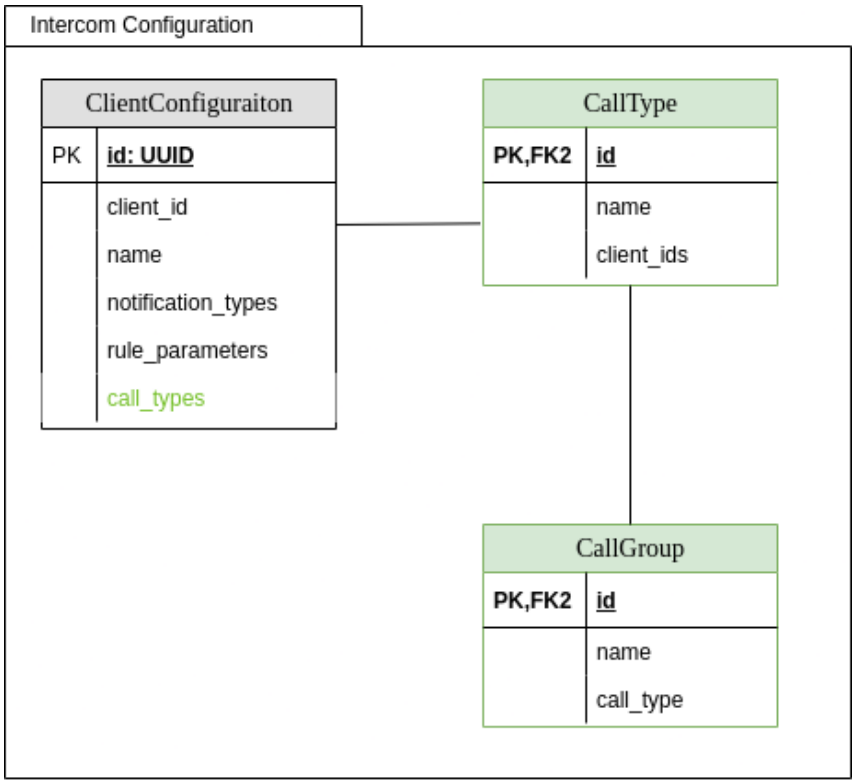


Abbildung 5.6: ERD Ausschnitt - Konfiguration Gegensprechanalge

Verfügbarkeit und Registrierung

Um Sprachverbindungen zwischen Clients aufzubauen, müssen diese Nachrichten austauschen können. Die Verbindung wird durch den Sender mit einem Offer initialisiert. Dieses muss an den Empfänger übermittelt werden. Dieser antwortet schliesslich mit einer Answer, welche an den Sender übermittelt werden ist. Bevor die Verbindung aufgebaut ist, kennen die beiden Clients sich gegenseitig noch nicht. Es braucht deshalb eine Instanz, welche beide Seiten kennt und die Übermittlung der Daten übernehmen kann. Bei Praxisruf fällt diese Aufgabe dem Cloud Service zu. Um dies zu ermöglichen wird Cloud Service umeine Schnittstelle erweitert, die es ermöglicht langlebige Verbindungen zu öffnen und registrieren. Sobald ein Client sich angemeldet hat, baut er eine Verbindung zum CloudService auf. Beim Verbindungsaufbau gibt der Client seine technische Identifikation mit. Der Cloud Service kann damit intern eine der verfügbaren Verbindungen und den dazugehörigen Identifikatoren führen. Diese Liste kann beim Verbindungsaufbau verwendet werden, um die Offers und Answers an die jeweiligen Clients zu übermitteln.

Dieses Konzept wird im intercom Modul des Cloud Service umgesetzt. Um die Funktionalität zu ermöglichen werden zwei Komponenten benötigt. Diese kapseln die Funktionalität die unabhängig von der Technologie zum Verbindungsaufbau notwendig ist. Um sicherzustellen, dass diese Unabhägigkeit bleibt, werden hier die Interfaces für diese beiden Komponenten spezifiziert. Es braucht erstens eine Komponente, welche Verbindungen etabliert und Nachrichten über diese Verbindungen senden kann. Diese Funktionalität wird mit der Komponente ClientConnector umgesetzt.

```
1  /**
2   * Contracts for clients to register an intercom connection in Praxisruf
3   *
4   * Once a connection is established it can be used, to negotiate messages between
5   * registered clients. This enables signaling server functionality for when
6   * establishing Peer To Peer Connections between clients.
7   *
8   * @param <T> Type of the connection
9   * @param <M> Type of messages that will be exchanged
10  */
11 public interface ClientConnector<T, M> {
12
13     /**
14      * Receives a message and forwards it to all relevant registered connections.
15      * M is expected to contain the key any relevant connection.
16      */
17     void sendMessage(M message);
18
19     /**
20      * Is called after a connection has been established. The established connection
21      * is stored in a ConnectionRegistry with key: clientId and value: connection.
22      */
23     void afterConnectionEstablished(UUID clientId, T connection);
24
25     /**
26      * Is called after a connection has been closed. The closed connection is
27      * removed from the ConnectionRegistry.
28      *
29      * @param connection
30      */
31     void afterConnectionClosed(T connection);
32 }
```

Listing 5: ClientConnector.java

Weiter braucht es eine Komponente, welche Buch führt über bekannte Verbindungen. Diese muss Verbindungen anhand einer id registrieren und Verbindungen wieder entfernen können.

```
1  /**
2   * Contracts for managing connections created by ClientConnector.
3   * @param <T> Type of the connections
4   */
5  public interface ConnectionRegistry<T> {
6
7      /**
8       * Registers the given connection in a key value store using clientId as key.
9       * @return boolean - whether the registration is registered
10      */
11     boolean register(UUID clientId, T connection);
12
13     /**
14      * Removes the given connection from the key value store
15      * @return boolean - whether the registration is unregistered
16      */
17     boolean unregister(T connection);
18 }
```

Listing 6: ClientConnector.java

Der Ablauf von Anmeldung und Registrierung funktioniert damit grundsätzlich wie bisher. Er wird aber um eine zusätzliche Registrierung über eine permanente Verbindung zum Intercom Modul ergänzt.

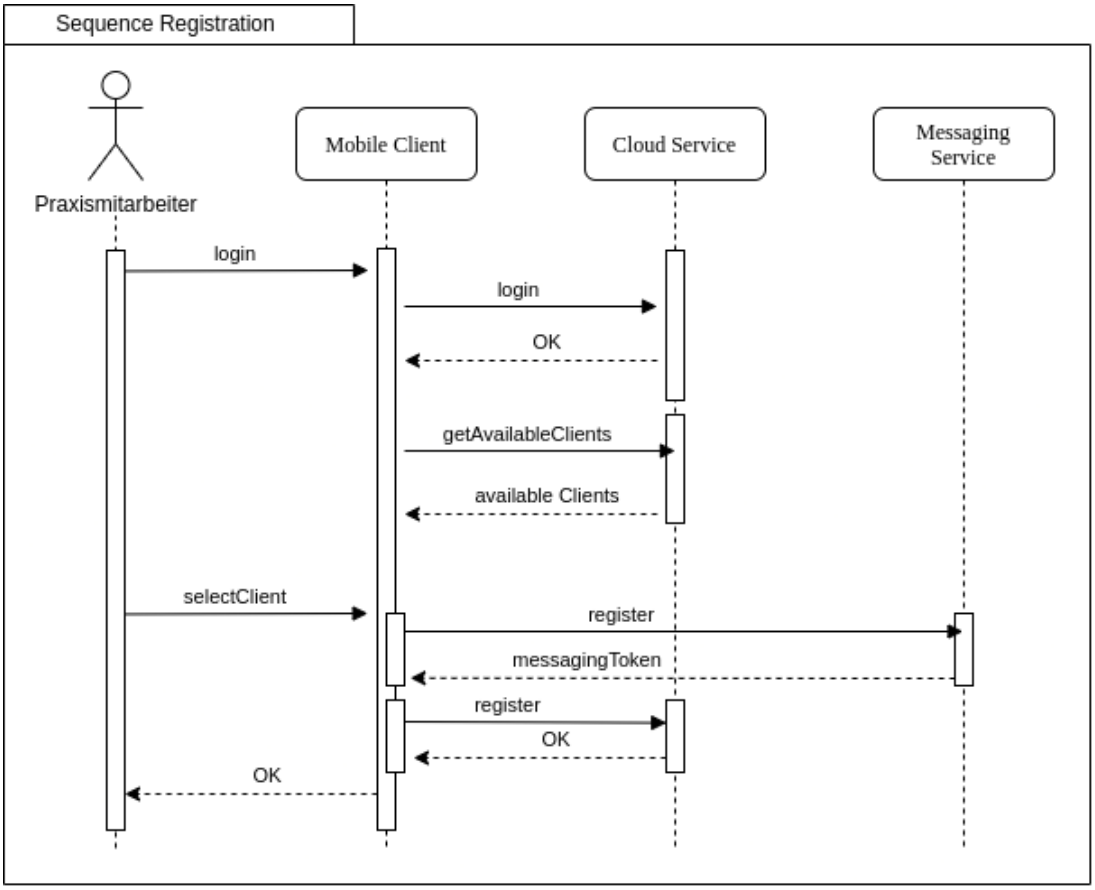


Abbildung 5.7: Mockup Home

Verbindungsaufbau

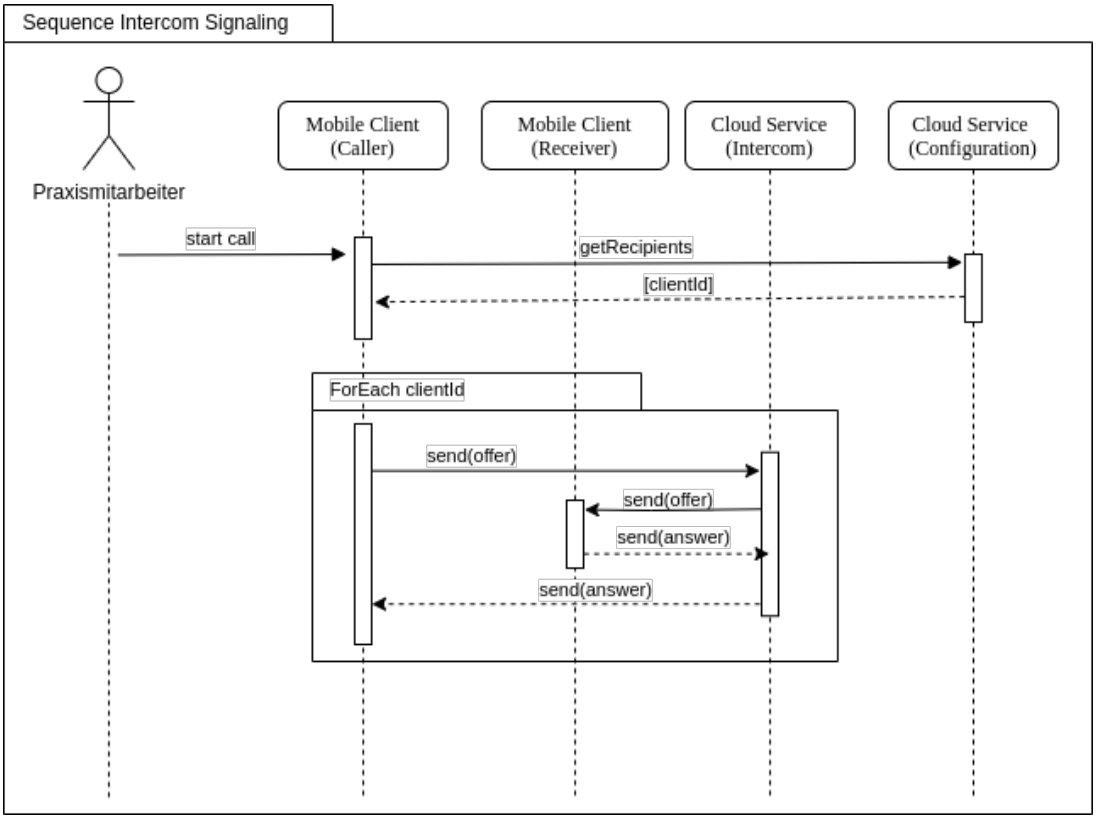


Abbildung 5.8: Ablauf Verbindungsaufbau Gegensprechanalge

Wenn der Praxismitarbeitende im Mobile Client einen Button der Gegensprechanlage tippt, soll eine Sprachverbindung zu anderen Clients aufgebaut werden. Zum Zeitpunkt an dem der Button getippt wird, weiss der Mobile Client nicht, zu welchen Clients diese Verbindung aufgebaut werden soll. Als erstes muss deshalb beim Cloud Service angefragt werden, welche Clients mit dem betätigten Button angesprochen werden sollen. Der Cloud Service bietet dazu einen Endpoint an über den die technischen Identifikatoren die in der CallGroup des verwendeten Buttons hinterlegt sind geladen werden.

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 Cloud Service ausgetauscht werden. Alle verfügbaren Clients haben sich bei der Anmeldung mit dem Cloud Service verbunden. Der Cloud Service 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 Cloud Service. Dieses Anfrage beinhaltet das ICE Offer und die Client Ids des von Ausgangs- und Ziel-Client. Der Cloud Service 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.

Unterhaltung

Mute Button End Call Button

Verbindungsende

Caller nimmt Finger vom Button. Receiver hat button zum abbrechen.

5.5 Zusammenfassung

Domänenmodell Cloud Service

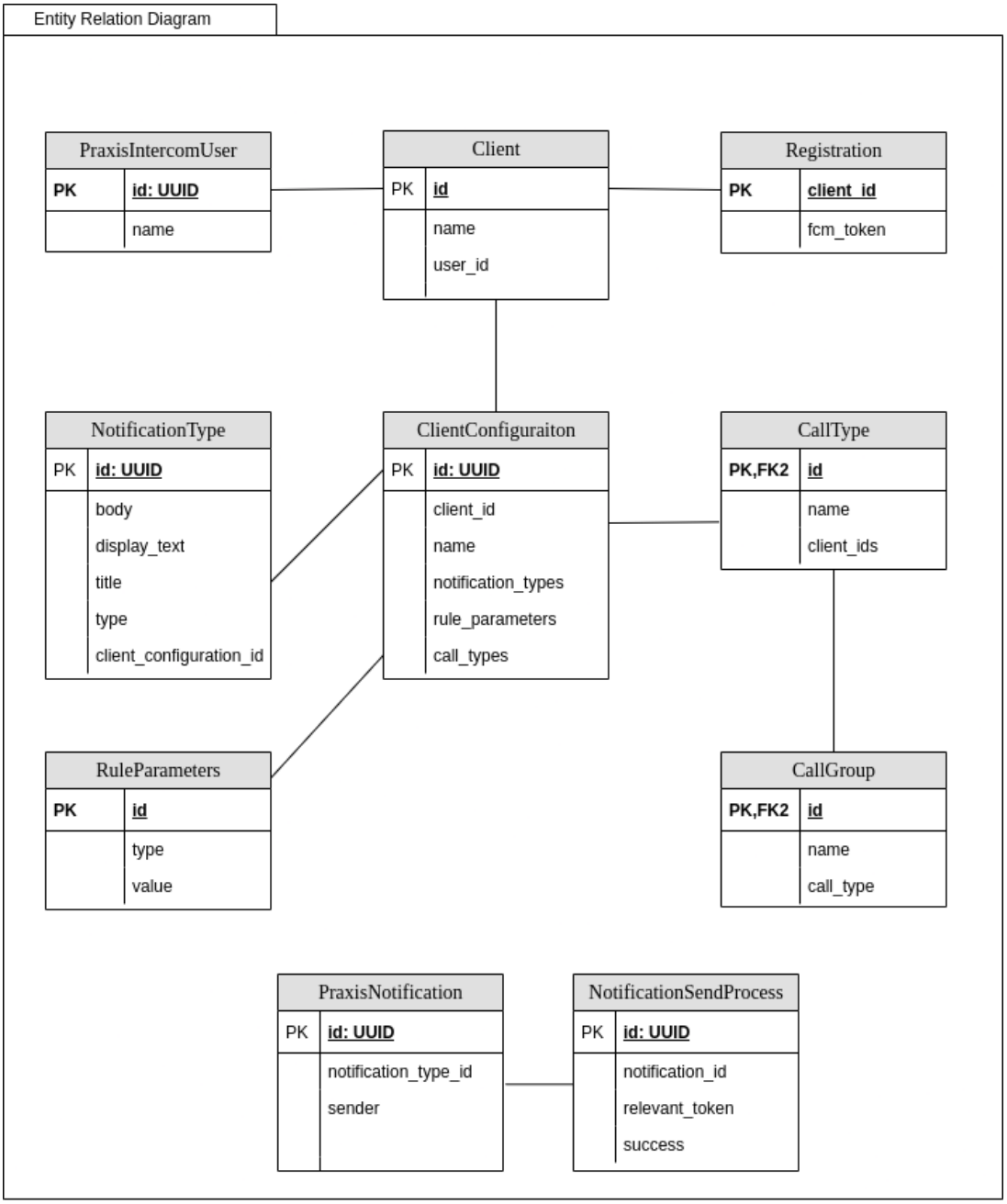


Abbildung 5.9: Entitiy Relation Diagramm - Cloud Service

API Cloud Service

Der Cloud Service 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

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

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

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

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

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

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

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

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

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

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

Abbildungsverzeichnis

A Aufgabenstellung

Fachhochschule Nordwestschweiz

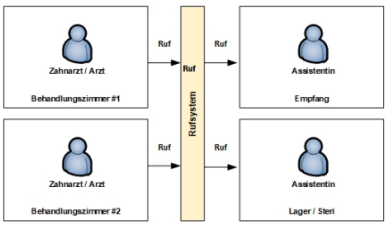
App
Web

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

[4] 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 .....