

Cloudbasiertes Praxisrufsystem

IP 5 - Technischer Bericht

20. August 2021

Studenten Joshua Villing, Kevin Zellweger

Fachbetreuer Daniel Jossen

Auftraggeber Daniel Jossen

Studiengang Informatik

Hochschule Hochschule für Technik

Management Summary

Ärzte und Zahnärzte haben den Anspruch in Ihren Praxen ein Rufsystem einzusetzen. Dieses Rufsystem ermöglicht, dass der behandelnde Arzt über einen Knopfdruck Hilfe anfordern oder Behandlungsmaterial bestellen kann. Heute kommerziell erhältlichen Rufsysteme beruhen meistens auf proprietären Standards und veralteten Technologien.[1] Die Problemstellung für dieses Projekt besteht darin, ein cloudbasiertes, leicht konfigurierbares und erweiterbares Rufsystem umzusetzen.

In diesem Projekt wurde ein Konzept für das Praxisrufsystem “Praxisruf” erarbeitet und umgesetzt. Praxisruf ist ein Rufsystem, welches das Versenden von Benachrichtigungen in einer Praxis ermöglicht. Das umgesetzte System wird von den Endbenutzern mit einer App auf einem Android Tablet oder IPad benutzt. Die dazu entwickelte Anwendung erlaubt es den Benutzern vorkonfigurierte Benachrichtigungen zu versenden und empfangen. Welche Benachrichtigungen versendet und empfangen werden, kann von Administratoren über eine Web-Oberfläche konfiguriert werden. Der Funktionsumfang des umgesetzten Rufsystems beschränkt sich auch das Versenden und Empfangen von Benachrichtigungen. Die Funktionen Gegensprechanlage und Text To Speech für Benachrichtigung wurden im Rahmen dieses Projekts nicht behandelt.

Nach Projektabschluss sehen wir nun drei Optionen für das weitere Vorgehen:

1. Praxisruf einsetzen

Praxisruf könnte bereits heute als produktives Benachrichtigungssystem eingesetzt werden. Im Gegensatz zu vielen bestehenden Systemen kann Praxisruf allerdings nicht als Gegensprechanlage verwendet werden. Weiter erlaubt die Konfiguration der Übermittlung von Benachrichtigungen erst sehr einfache Regeln. Zur einfacheren Konfigurierbarkeit sollte dieses Regelwerk erst noch erweitert werden. Aufgrund dieser Einschränkungen raten wir davon ab, Praxisruf in einem produktiven Umfeld einzusetzen.

2. Praxisruf weiterentwickeln

Praxisruf könnte um die Funktionen Gegensprechanlage und Text To Speech erweitert werden. Weiter könnten die verfügbaren Regeln zum Empfangen von Benachrichtigungen ausgebaut werden. Praxisruf wurde konzipiert, um leicht erweiterbar zu sein. Dementsprechend können diese Änderungen mit verhältnismässig geringem Aufwand eingebaut werden. Eine mögliche Gefahr für die Weiterentwicklung ist dabei die bestehende Mobile Applikation. Unsere Erfahrung mit der dafür gewählten Technologie zeigt, dass viele Funktionen nicht für alle Plattformen funktionieren oder Einschränkungen bezüglich Kompatibilität bringen. Wir raten deshalb Praxisruf weiterzuentwickeln, aber die dazugehörige App nicht in der aktuellen Form weiterzuverwenden.

3. Praxisruf weiterentwickeln (nativ)

Praxisruf könnte wie in Option 2 beschrieben weiterentwickelt werden. Um Probleme bei der Appentwicklung zu minimieren, kann der Mobile Client als native Applikation neu entwickelt werden. Nach dem initialen Aufwand der Migration würde dies die Applikation deutlich zukunftssicherer machen. Der Wartungsaufwand wird nach unserer Einschätzung dadurch nicht wachsen. Der zusätzliche Aufwand zweier Applikationen zu warten wird dadurch aufgehoben, dass die Anbindung von Gerätehardware und betriebssystemnahen Funktionen deutlich besser unterstützt ist.

Empfehlung

Wir empfehlen mit der Option 3 “Praxisruf weiterentwickeln (nativ)” fortzufahren. Der Mobile Client sollte neu als native Applikation umgesetzt werden, um bessere Kompatibilität zu gewähren. Das Praxisrufsystem sollte zudem um zusätzliche Konfigurationsoptionen und die Funktionen Gegensprechanlage erweitert werden. Sobald Praxisruf auch diese Funktion unterstützt bietet es einen breiteren Funktionsumfang als bestehende Lösungen. Dabei ist es aber leicht konfigurierbar und kann somit für die konkreten Bedürfnisse jedes Kunden individualisiert werden.

Inhaltsverzeichnis

1 Einleitung	1
2 Vorgehensweise	2
2.1 Projektplan	2
2.2 Meilensteine	3
3 Anforderungen	4
4 Evaluation Technologien	6
4.1 Mobile Client Evaluation	6
4.2 Messaging Service	7
4.3 Cloud Service	8
5 Konzept	9
5.1 Systemarchitektur	9
5.2 Mobile Client	10
5.3 Cloud Service	16
5.4 Admin UI	28
5.5 Proof Of Concept	29
6 Umsetzung	32
6.1 Resultate	32
6.2 Tests	38
6.3 Fazit	40
7 Schluss	42
Literaturverzeichnis	43
Abbildungsverzeichnis	45
A Aufgabenstellung	46
B Quellcode	47
C Benutzerhandbuch	48
D Installationsanleitung	50

E Features und Testszenarien	53
F Ehrlichkeitserklärung	57

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. Eine vom Kunden 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 Tablets kann die zu entwickelnde App installiert und betrieben werden. Das Projekt deckt dabei die folgenden Ziele ab:

- Definition und Entwicklung einer skalierbaren Softwarearchitektur
- Die App ist auf Android und iOS basierten Tablets einsetzbar
- Die App besitzt eine integrierte Gegesprechfunktion (1:1 oder 1:m Kommunikation)
- Auf der App können beliebige Buttons konfiguriert werden, die anschliessend auf den anderen Tablets einen Alarm oder eine Meldung (Text- oder Sprachmeldung) generieren
- Textnachrichten können als Sprachnachricht (Text to Speech) ausgegeben werden
- Verschlüsselte Übertragung aller Meldungen zwischen den einzelnen Stationen
- Integration der Lösung in den Zahnarztbehandlungsstuhl über einen Raspberry PI
- Offene API für Integration in Praxisadministrationssystem

Die Hauptproblemstellung dieser Arbeit ist die sichere und effiziente Übertragung von Sprach- und Textmeldungen zwischen den einzelnen Tablets. Dabei soll es möglich sein, dass die App einen Unicast, Broadcast und Multicast Übertragung der Daten ermöglicht. Über eine offene Systemarchitektur müssen die Kommunikationsbuttons in der App frei konfiguriert und parametrisiert werden können.”¹ [1]

Bei Projektstart bestehen zwei potenzielle Gefahren. Keiner der Projektteilnehmenden hat vorgängige Erfahrung mit der Entwicklung von mobilen Applikationen und der Verwendung von Amazon Webservices. Dementsprechend ist mit Mehraufwand in diesen Bereichen zu rechnen. Weiter besteht eine Gefahr in der pandemischen Situation bei Projektstart im Frühling 2021. Die Organisation und Kommunikation des Projektes wird auf die Einschränkungen der aktuellen Lage angepasst und von Anfang ausschliesslich über digitale Werkzeuge organisiert.

Diese Projektarbeit soll in vier Phasen umgesetzt werden. In der ersten Phase werden die grundlegenden Anforderungen als Meilensteine besprochen und priorisiert. Dazu werden oberflächliche Konzepte erarbeitet. In der zweiten Phase werden Technologien evaluiert, die verwendet werden, um die wichtigsten Anforderungen umzusetzen. Es wird ein Proof Of Concept implementiert, um sicherzustellen, dass die technischen Voraussetzungen gegeben sind alle Anforderungen umzusetzen. In der dritten Phase wird auf Basis des Proof Of Concept das Praxisrufsystem “Praxisruf” umgesetzt. Die detaillierten Anforderungen werden dabei in einem iterativen Verfahren zusammen mit dem Kunden erarbeitet.

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.

¹Ausgangslage, Ziele und Problemstellung im Originaltext der Aufgabenstellung

2 Vorgehensweise

2.1 Projektplan

Übersicht



Abbildung 2.1: Projektplan

Das Projekt “Cloudbasiertes Praxisrufsystem” soll in vier Phasen umgesetzt werden. In der ersten Phase (KW7-KW13) wird die organisatorische Infrastruktur für das Projekt aufgebaut. Es werden Grobkonzepte für das umzusetzende System erfasst. Dazu werden Technologien evaluiert, die zur Umsetzung verwendet werden können und die wichtigsten Ziele des Projektes als Meilensteine festgehalten. Im Fokus der zweiten Phase (KW13-KW17) steht das Umsetzen eines Proof Of Concepts. Dieser soll die Gültigkeit der Resultate aus der ersten Phase verifizieren. Bei Bedarf werden Anpassungen an den gewählten Technologien und erstellten Grobkonzepten gemacht. Die Architektur und das Konzept werden weiter verfeinert und erste Anforderungen werden konkretisiert. Die Umsetzung des Systems findet schliesslich in der dritten Phase (KW17-KW38) statt. Dazu gehört es, laufend die als nächstes umzusetzenden Anforderungen im Detail zu definieren und die Konzepte dafür zu erweitern. Die vierte Phase (KW14-KW32) läuft parallel zu der dritten Phase und beschäftigt sich mit dem Testen des Systems. Automatisierte Unit und Smoke Tests sollen während der ganzen Projektdauer für die entwickelten Komponenten gemacht werden. Zum Ende des Projekts soll das System zudem mit dem Benutzer getestet werden.

2.2 Meilensteine

In der Anfangsphase des Projektes wurden folgende Meilensteine definiert:

Id	Beschreibung
M01	Proof Of Concept - Setup Ein Mobile Client kann auf iOS installiert werden und mit einem Backendservice kommunizieren.
M02	Proof Of Concept - Messaging Ein Mobile Client kann Benachrichtigungen empfangen und Push-Benachrichtigungen anzeigen.
M03	Versenden mit Registration Mobile Clients können sich beim Praxisrufsystem registrieren und Benachrichtigungen empfangen. Alle registrierten Clients erhalten alle Benachrichtigungen.
M04	Benachrichtigungen konfigurierbar Der Mobile Client lädt seine Konfiguration vom Backend und zeigt dynamisch konfigurierte Buttons an über die Benachrichtigungen versendet werden.
M05	Benutzeroberfläche für Konfiguration Ein Administrator kann das System über eine Benutzeroberfläche konfigurieren.
M06	1:N Versenden Konfigurierbar Die Konfigurationsmöglichkeiten werden erweitert. Es kann konfiguriert werden, welcher Client sich für welche Benachrichtigungen interessiert. Alle Clients erhalten nur für sie relevante Benachrichtigungen.
M07	Text to Speech Empfangene Benachrichtigungen können dem Benutzer vorgelesen werden.
M08	Raspberry Pi Anbindung Benachrichtigungen können über ein Raspberry Pi mit angeschlossenem Button versendet werden.
M09	Sprachkommunikation 1:1 Der Mobile Client unterstützt direkte Anrufe zwischen zwei Geräten.
M10	Sprachkommunikation 1:n Der Mobile Client unterstützt Gruppenanrufe mit mehreren Geräten gleichzeitig.

3 Anforderungen

Die umzusetzenden Anforderungen wurden während des Projektes iterativ zusammen mit dem Kunden erarbeitet. Dieses Kapitel zeigt den Stand der erarbeiteten Anforderungen bei Projektabschluss. Alle Anforderungen wurden zuerst aus fachlicher Sicht mit User Stories festgehalten. Jede User Story beschreibt ein konkretes Bedürfnis der Benutzer.

Die erfassten User Stories werden in drei Bereiche unterteilt. Im ersten Bereich werden Bedürfnisse von Praxismitarbeitenden festgehalten. Praxismitarbeitende verwenden die mobile Applikation von Praxisruf, um Benachrichtigungen zu versenden und empfangen. Im zweiten Bereich werden die Bedürfnisse von Praxisverantwortlichen beschrieben. Diese Benutzergruppe ist dafür verantwortlich, Praxisruf für Praxismitarbeitende zu konfigurieren. In einem dritten Bereich werden User Stories aus Sicht des Kunden definiert, welche die Rahmenbedingungen und Bedürfnisse des Auftraggebers adressieren.

Aufgrund der User Stories werden Features definiert, welche konkrete testbare Szenarien und die erwarteten Ergebnisse definieren.²

Praxismitarbeitende

Id	Anforderung	Feature
U01	Als Praxismitarbeiter/-in möchte ich Benachrichtigungen versenden können, damit ich andere Mitarbeitende über Probleme und Anfragen informieren kann.	F01
U02	Als Praxismitarbeiter/-in möchte ich Benachrichtigungen empfangen können, damit ich auf Probleme und Anfragen anderer Mitarbeitenden reagieren kann.	F02
U03	Als Praxismitarbeiter/-in möchte ich nur Benachrichtigungen sehen, die für mich relevant sind, damit ich meine Arbeit effizient gestalten kann.	F02
U04	Als Praxismitarbeiter/-in möchte ich über empfangene Benachrichtigungen aufmerksam gemacht werden, damit ich keine Benachrichtigungen verpasse.	F04
U05	Als Praxismitarbeiter/-in möchte ich sehen welche Benachrichtigungen ich verpasst habe, damit ich auf verpasste Benachrichtigungen reagieren kann.	F04
U06	Als Praxismitarbeiter/-in möchte ich eine Rückmeldung erhalten, wenn eine Benachrichtigung nicht versendet werden kann, damit Benachrichtigungen nicht verloren gehen.	F03
U07	Als Praxismitarbeiter/-in möchte ich auswählen können an welchem Gerät ich das Praxisrufsystem verwende und die dafür erstellte Konfiguration erhalten, damit das Praxisrufsystem optimal verwendet werden kann.	F05
U08	Als Praxismitarbeiter/-in möchte ich einen physischen Knopf am Behandlungsstuhl haben damit ich Nachrichten darüber versenden kann.	F07
U09	Als Praxismitarbeiter/-in möchte ich, dass mir Benachrichtigungen vorgelesen werden, damit ich informiert werde, ohne meine Arbeit unterbrechen zu müssen.	F08
U10	Als Praxismitarbeiter/-in möchte ich mit einem anderen Client Unterhaltungen führen können, damit Fragen direkt geklärt werden können.	F09
U11	Als Praxismitarbeiter/-in möchte ich Unterhaltungen mit mehreren anderen Clients gleichzeitig führen können, damit komplexe Fragen direkt geklärt werden können.	F10

²Siehe Anhang E - Features und Testszenarien

Praxisverantwortliche

Id	Anforderung	Features
U12	Als Praxisverantwortliche/-r möchte ich mehrere Geräte verwalten können, damit das Praxisrufsystem in mehreren Zimmern gleichzeitig genutzt werden kann.	F06
U13	Als Praxisverantwortliche/-r möchte ich definieren können, welches Gerät welche Anfragen versenden kann, damit jedes Gerät für Anwendungsbereich optimiert ist.	F06
U14	Als Praxisverantwortliche/-r möchte ich die Konfiguration des Praxisrufsystems zentral verwalten können, damit das Praxisrufsystem für die Anwender optimiert werden kann.	F06
U15	Als Praxisverantwortliche/-r möchte ich definieren können, welche Geräte untereinander eine Sprachverbindung aufbauen können, damit alle Mitarbeitenden ihrer Funktion entsprechend die Gegensprechanlage nutzen können.	F06
U16	Als Praxisverantwortliche/-r möchte ich definieren können, welche Benachrichtigung über einen physischen Knopf am Behandlungsstuhl versendet wird damit der Knopf für die Mitarbeitenden optimiert ist.	F06

Auftraggeber

Id	Anforderung	Features
T01	Als Auftraggeber möchte ich, dass das Praxisrufsystem über iPads bedient werden kann, damit ich von bestehender Infrastruktur profitieren kann.	n/A
T02	Als Auftraggeber möchte ich, dass das Praxisrufsystem über Android Tablets bedient werden kann, damit es in Zukunft für eine weitere Zielgruppe verwendet werden kann.	n/A
T03	Als Auftraggeber möchte ich, dass die Codebasis für das Praxisrufsystem für Android und iOS verwendet werden kann, damit ich die Weiterentwicklung optimieren kann.	n/A
T04	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.	n/A

4 Evaluation Technologien

4.1 Mobile Client Evaluation

Da keiner der Projektteilnehmer Erfahrungen im Bereich der mobilen App-Entwicklung mitbringt, wurde zuerst eine Evaluation der verfügbaren Frameworks durchgeführt. Dabei waren folgende Anforderungen das Hauptkriterium:

- Es muss eine geteilte Codebasis für iOS und Android geben.
- Zugang zu Gerätehardware, insbesondere Mikrofon und Lautsprecher muss möglich sein.
- Unterstützung für iOS muss gegeben sein.

Kotlin Multiplatform

Kotlin ist eine auf der Java Virtual Machine basierende Sprache, welche von JetBrains entwickelt wurde. Kotlin Multiplatform Mobile ist ein Framework, dass es erlaubt allgemeine Business-Logik in purrem Kotlin zu schreiben und auf iOS sowie Android zu verwenden. Native Funktionen des Betriebssystems müssen dabei über separate APIs verwendet werden.[2] Viele Anforderungen an den Mobile-Client benötigen native Funktionen. Der Anteil an Business-Logik ist dabei eher gering. Dies führt dazu, dass ein grosser Teil des Mobile-Clients zweimal entwickelt werden müsste.

Progressive Web App

Eine Progressive Web App (PWA) wird grundsätzlich wie eine reguläre Webapplikation entwickelt. Daraus ist garantiert nur eine Codebase für alle Plattformen notwendig. Ziel einer PWA ist es so nahe wie möglich an eine Native App heranzukommen hinsichtlich Usability, User-Experience und Performance.[3] Ein Testprojekt im Rahmen der Evaluation hat gezeigt, dass Push-Benachrichtigungen auf Android Geräten problemlos funktionieren. Unter iOS jedoch kann eine PWA nur mittels Safari installiert werden. Safari erlaubt es allerdings nicht Push-Meldungen zu empfangen.[4], [5]

Cordova

Cordova ist ein Framework von Apache mit dem Cross-Platform Apps auf der Basis von standard Web-Technologien erstellt werden können. Der Zugriff auf native Funktionen des Betriebssystems wird mittels Plugins gelöst. Die Entwicklung für mehrere Plattformen benötigt weiterhin unterschiedliche Entwicklungsumgebungen (Android Studio / Xcode). Die Codebasis für alle Plattformen ist dabei aber dieselbe. Weiter bietet Cordova einen Plattformübergreifenden CLI-Workflow, sodass es für den Entwickler keine Umstellung im Prozess gibt.[6] Dadurch das Cordova auf einen Wrapper setzt und die Plugins für die Device-Apis abgesetzt werden, ist die App als Gesamtes aber nicht rein nativ und erreicht daher auch nicht die volle native Performance.[7]

Native Script

Nativescript baut ähnlich wie Cordova und PWAs auf standard Web-Technologien auf. Der Hauptunterschied liegt darin, dass Nativescript die Plattformspezifischen APIs direkt abstrahiert. So wird eine entwickelte Applikation plattformübergreifend zu einer rein nativen App kompiliert. Als Projekt der Open-JS Foundation hat NativeScript eine sehr breite Community die fortlaufend neue Features entwickelt. Besonders beliebt ist, dass Nativescript mit den gängigsten Frameworks VueJS, Angular und React verwendet werden kann. Ähnlich wie bei Cordova können alle benötigten Betriebssystemfunktionen über Plugins integriert werden.[7], [8]

Flutter

Flutter ist ein Toolkit von Google um native Cross-Platform Apps zu entwickeln. Die Implementierung erfolgt in der ebenfalls von Google supporteten Sprache Dart.[9] Der Ursprung von Google führt dazu, dass Flutter primär für Android konzipiert ist. Die UI-Komponenten werden nicht vom Betriebssystem verwendet, sondern selber designt. Als Konsequenz daraus, falls ein natives “look and feel” gewünscht ist, müssten diese Komponenten separat entwickelt werden.[10]

Ergebnis

Die Evaluation der Mobile Technologien hat gezeigt, dass eine PWA grundsätzlich nicht geeignet ist ein Praxisrufsystem umzusetzen. Serverseitige Push-Benachrichtigungen sind für das Praxisrufsystem zwingend notwendig, auf iOS und Safari aber nicht möglich. Kotlin und Flutter benötigen trotz einer geteilten Codebasis viel plattformspezifische Entwicklungen und bauen gleichzeitig auf Technologien auf, mit denen wir nur wenig Erfahrung haben. Cordova und Nativescript bauen auf uns bereits bekannten Web-Technologien auf. Nativescript hat dabei den Vorteil, dass es die tatsächlich nativen APIs abstrahiert und am Ende native Applikationen für die jeweiligen Plattformen generieren kann. Wir haben uns schlussendlich entschieden den Mobile-Client in NativeScript zu entwickeln, da dies die tatsächlich Nativen APIs abstrahiert und so am besten geeignet für unsere Anforderungen scheint.

Die Evaluation hat weiter gezeigt, dass die Entwicklung des Mobile Clients zwingend MacOS Hardware mit einer XCode Entwicklungsumgebung benötigt, da dies zwingend Notwendig ist, um eine Applikation während der Entwicklung auf iOS zu installieren.[11] Zudem können die notwendigen nativen Funktionen von iOS nur zusammen mit einem kostenpflichtigen Apple Developer Account verwendet werden.[12]

4.2 Messaging Service

Für das Praxisrufsystem wird ein Dienst benötigt, mit dem Benachrichtigungen an eine Mobile Applikation gesendet werden können. Als Reaktion auf eine empfangene Benachrichtigung muss es möglich sein, Push-Benachrichtigungen auf dem Gerät anzuzeigen. Die Technologie, die dazu verwendet wird, muss dabei mit der für den Mobile Client gewählten Technologie kompatibel sein. Für den Mobile Client wurde das Native Script Framework ausgewählt. Damit sind Push-Benachrichtigungen auf iOS und Android möglich. Push-Benachrichtigungen auf iOS sind allerdings nur über die Anbindung von Firebase Cloud Messaging (FCM) möglich.[13]

Bevor Push-Benachrichtigungen auf dem Gerät angezeigt werden können, muss die entsprechende Benachrichtigung an den Mobile Client gesendet werden. Mit Event-Sources[14] ist es möglich, Anfragen von der Serverseite an einen Mobile Client zu senden. Dies hat allerdings den Nachteil, dass eine Konstante HTTP Verbindung zwischen Client und Server bestehen muss. Alternativ kann ein Message Broker verwendet werden, um Benachrichtigungen an den Mobile Client zu senden.[15] Beide Optionen haben die Einschränkung, dass sie es nicht ermöglichen Push-Benachrichtigungen auf dem Gerät anzuzeigen. Um dies auf iOS mit Nativescript zu ermöglichen, müsste zusätzlich ein Firebase Cloud Messaging Service angebunden werden. Firebase Cloud Messaging ermöglicht es dabei selbst von einem Server Umfeld Benachrichtigungen an Endgeräte zu versenden.[16] Firebase Cloud Messaging unterstützt damit die Anforderungen, von einem Cloud Server Benachrichtigungen an einen Mobile Client zu senden und Push-Benachrichtigungen auf dem Gerät anzuzeigen. In der Folge wird für das Praxisrufsystem ausschliessliche Firebase Cloud Messaging als Messaging Service verwendet.

4.3 Cloud Service

Backendservices die für das Praxisrufsystem nötig sind, werden mit Java und Spring Boot umgesetzt. Spring Boot vereinfacht Grundlagenarbeit, die zum Aufsetzen eines Backendservices nötig sind und bietet gleichzeitig die Werkzeuge die nötig sind um sichere, skalierbare Microservices zu implementieren.[17] Spring und Java sind zudem Technologien, mit welchen alle Projektteilnehmer bereits Erfahrung gesammelt haben. Diese für die Umsetzung in diesem Projekt zu verwenden, ermöglicht es die nötigen Services noch effizienter umzusetzen und den Fokus auf das technische Konzept sowie die Anforderungsanalyse zu legen.

In der Anforderung T04³ ist vorgegeben, dass der Betrieb aller Cloud- und Web-Applikationen mit Amazon Webservices erfolgen muss. Mit Elastic Beanstalk von AWS ist die der Betrieb von Java Applikationen auf AWS möglich.[18]

³Siehe Kapitel 3

5 Konzept

5.1 Systemarchitektur

Abgrenzung

Dieses Kapitel gibt eine Übersicht über die Systemarchitektur als Ganzes. Die Architektur beschränkt sich dabei auf die Anforderungen, die innerhalb des Projektrahmens umgesetzt wurden. Komponenten für Teile die ausserhalb des Projektumfangs gefallen sind, werden hier nicht behandelt.

Übersicht

Das cloudbasierte Praxisrufsystem wird in vier Komponenten unterteilt. Im Zentrum steht eine cloudbasierte Applikation (Cloud Service) welche es ermöglicht Konfigurationen persistent zu verwalten und das Versenden von Benachrichtigungen anhand dieser Konfigurationen koordiniert. Der Cloud Service benutzt einen externen Messaging Service zum Versenden von Benachrichtigungen. Dabei ist der Messaging Service lediglich für die Zustellung von Benachrichtigungen verantwortlich. Zur Verwaltung der Konfigurationen wird ein Web Frontend (Admin UI) erstellt. Dieses bietet einem Administrator die Möglichkeit Konfigurationen aus dem Cloud Service zu lesen, erstellen, bearbeiten und löschen. Die Konfigurationen die über Admin UI und Cloud Service erstellt wurden, werden von einem Mobile Client verwendet. Mit dem Mobile Client kann der Benutzer Benachrichtigungen an andere Mobile Clients senden. Welche Benachrichtigungen ein Mobile Client senden kann und an wen diese Benachrichtigungen zugestellt werden, wird anhand der Konfiguration aus dem Cloud Service bestimmt.

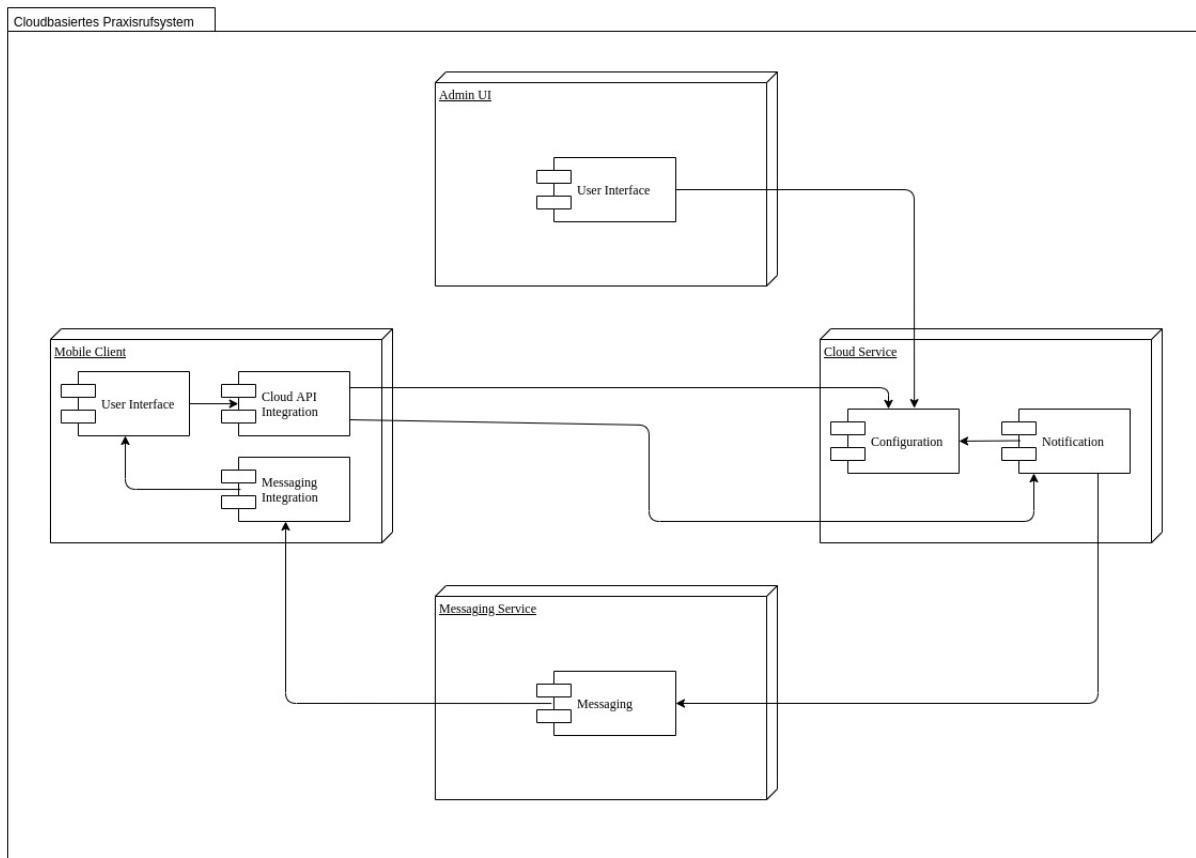


Abbildung 5.1: Systemkomponenten

5.2 Mobile Client

5.2.1 Framework Grundlagen

NativeScript bietet eine Abstraktion zu den nativen Plattformen Android und iOS. Die jeweilige NativeScript Runtime erlaubt es in Javascript (oder einem entsprechenden Application Framework) Code zu schreiben, welcher direkt für die entsprechende native Umgebung kompiliert wird [8].

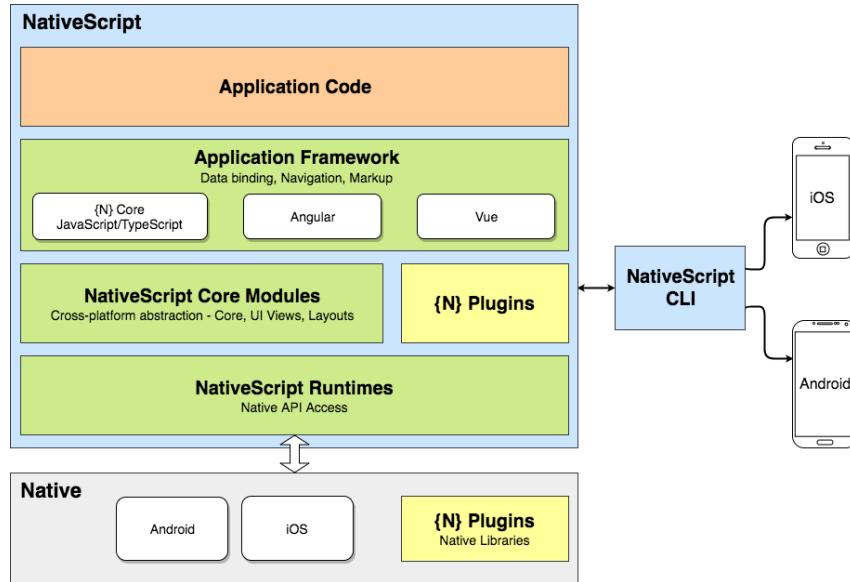


Abbildung 5.2: NativeScript-Overview
©OpenJS Foundation

Die Runtime agiert als Proxy zwischen Javascript und dem jeweiligen Ökosystem. Im Falle von iOS bedeutet dies, dass für alle Objective-C types ein JavaScript Prototype angeboten wird. Dies ermöglicht es direkt mit nativen Objekten zu interagieren. Im Umkehrschluss findet eine Typenkonversion via Marshalling Service statt[19].

5.2.2 Anwendung

Wir verwenden NativeScript Core als Framework des Mobile-Clients. In Kapitel *Mobile Client Evaluation* gehen wir auf die weiteren verfügbaren Frameworks ein und erläutern, weshalb wir uns gegen sie entschieden haben.

Die Client-Applikation ist in Module unterteilt. Ein Modul wird aus folgenden Komponenten definiert:

- UI-Markup: Statische Darstellung in XML
- Backend: Verhalten und Dynamisierung in Javascript
- Styling: Layout und Styles in CSS

Ein minimales Modul kann alleine aus einer XML-Datei bestehen. Die optionalen Javascript und CSS Dateien müssen denselben Namen haben wie die XML Datei, um vom Framework korrekt verknüpft zu werden. Dateien mit anderen Namen werden grundsätzlich vom Framework ignoriert. Natürlich steht es frei dennoch solche Dateien anzulegen und deren Funktionen zu verwenden z. B. als *Services* oder als *Code-Behind Komponenten*.

Zur Veranschaulichung der möglichen Interaktionen gehen wir auf die relevanten Aspekte des Home-Screen Modules ein.

Page Module

```
app
  └── home
      ├── home-page.xml
      ├── home-page.css
      ├── home-page.js
      └── home-model.js
```

home-page.xml deklariert die umgebenden Komponenten. Diese Komponenten stellen je nach Typ diverse Properties und Events zur Verfügung. Properties können entweder statisch befüllt oder aus dem Binding-Context geladen werden. Den Events können Callback-Functions zugewiesen werden. Es stehen alle Funktionen zur Verfügung, welche im Backendscript *home-page.js* exportiert werden.

```
1 <Page loaded="onPageLoaded" navigatingTo="onNavigatingTo"
2   xmlns="http://schemas.nativescript.org/tns.xsd"
3   xmlns:profile="components/profile">
4   <StackLayout id="profile" class="home-body">
5     <profile:profile-container/>
6     <StackLayout class="btn-box">
7       <Label textAlignment="center" text="Meldungen" class="section_title"/>
8       <GridLayout loaded="onGridLoaded" columns="auto, auto, auto" rows="auto
auto auto" horizontalAlignment="center">
9
10      </GridLayout>
11    </StackLayout>
12  </StackLayout>
13 </Page>
```

Listing 1: home-page.xml

Der Binding-Context ist ein JavaScript Objekt welches exklusiv im Page-Context zur Verfügung steht. Es ist allgemein Best-Practice dieses Objekt in einem eigenen Model zu verwalten. Das eigentliche Binding wird vom Backendscript *home-page.js* (Zeilen 8–11) während des ersten Ladens der Seite durchgeführt.

```
1 import {fromObject, Observable, ObservableArray} from '@nativescript/core'
2
3 export function HomeItemsViewModel() {
4   const viewModel = new Observable();
5   viewModel.notificationConfigurations = new ObservableArray([])
6
7   return viewModel
8 }
```

Listing 2: home-model.js

Das Backendscript ist für das dynamische Verhalten der Seite verantwortlich. Hier können die Interaktionen der Benutzer beliebig verarbeitet und der Binding-Context bei Bedarf verwaltet werden.

```
1 import {ApplicationSettings, Button, GridLayout} from "@nativescript/core";
2 import {MessageTrigger} from "~/components/message-trigger/message-trigger";
3 import {HomeItemsViewModel} from './home-model'
4 import {getClientConfiguration} from "~/services/configuration-api";
5 import {showError} from "~/error-dialog/error-dialog";
6
7 const model = new HomeItemsViewModel();
8 export function onPageLoaded(args) {
9   const mainComponent = args.object;
10  mainComponent.bindingContext = model;
11 }
```

```

12
13 export function onGridLoaded(args) {
14     const grid = args.object;
15     getClientConfiguration(ApplicationSettings.getString("clientId"))
16         .then(result => buildMessageUI(result, grid))
17         .catch((e) => showError("Loading Client Configuration failed", e, "OK"));
18 }
19
20 function buildMessageUI(clientConfiguration, grid) {
21     let rowCounter, columnCounter, rowIdx, columnIdx = 0;
22     clientConfiguration.map((conf) => {
23         const messageComp = new MessageTrigger(conf.title, conf.id);
24         grid.addChild(messageComp);
25         GridLayout.setRow(messageComp, rowIdx);
26         GridLayout.setColumn(messageComp, columnIdx);
27         rowCounter++;
28         columnCounter++;
29         if(columnCounter > 2) {
30             columnIdx = 0;
31             columnCounter = 0;
32         } else {
33             columnIdx++;
34         }
35         if(rowCounter > 3) {
36             rowIdx++;
37             rowCounter = 0;
38         } else {
39             rowCounter++;
40         }
41     })
42 }

```

Listing 3: home-page.js

Code-Behind Komponenten

Code-Behind Komponenten bieten die Möglichkeit zur Laufzeit dynamisch Grafikelemente dem UI hinzuzufügen. Komponenten die das Framework bereits zur Verfügung stellt können direkt mit `new <Component>()` instanziert werden. Bei Bedarf können diese Komponenten auch erweitert und mit zusätzlicher Funktionalität ausgestattet werden.

Da der Home-Screen dynamisch in Abhängigkeit der Client-Configuration erstellt werden muss, werden eigene `MessageTrigger` Komponenten verwendet.

Services

In Services werden diejenigen Funktionen ausgelagert, welche nicht direkt im Zusammenhang mit der grafischen Representation stehen. So z. B. die REST-Calls zur API.

5.2.3 Architektur

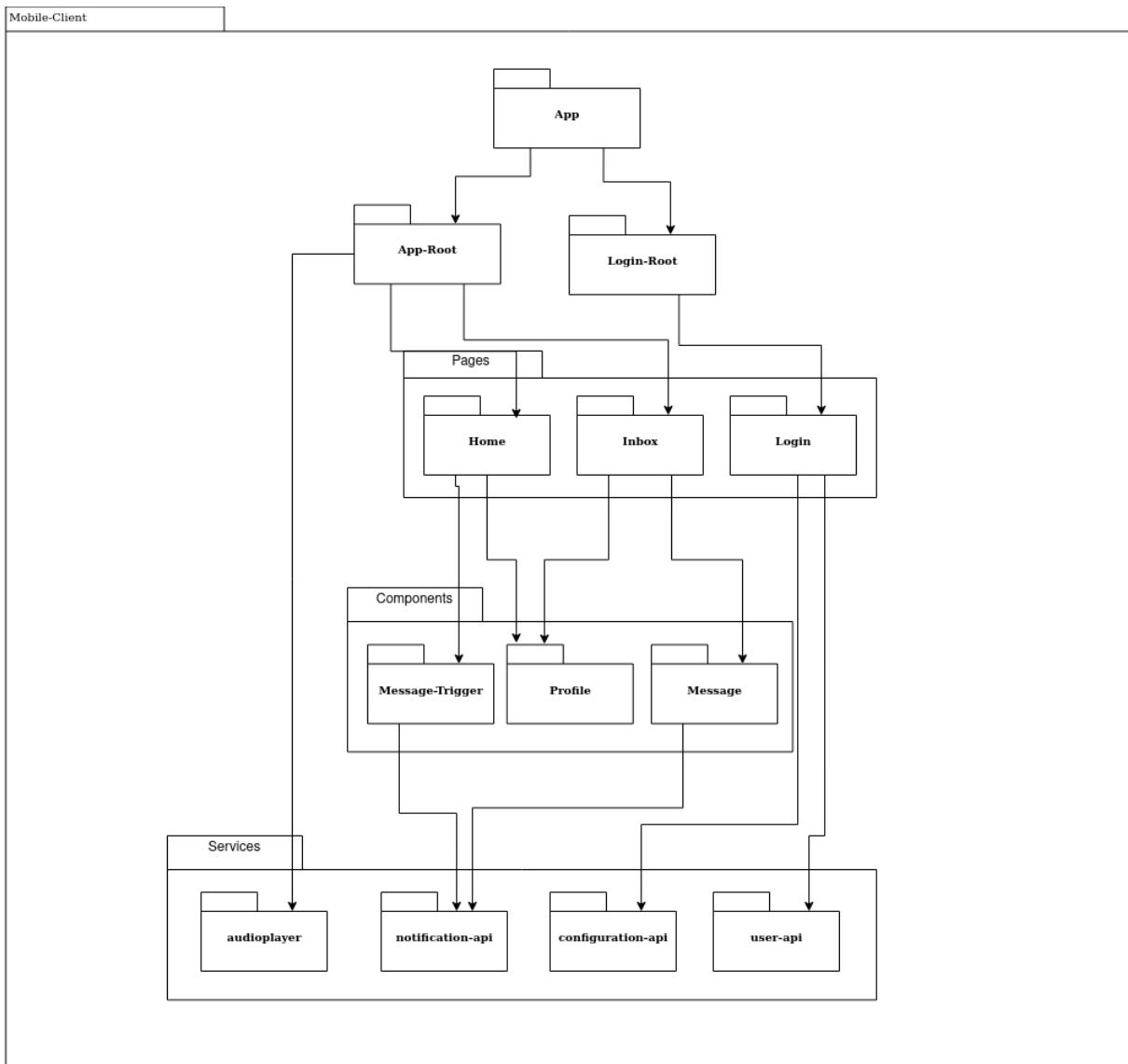


Abbildung 5.3: Mobile-Client Package Diagramm

Der Mobile-Client wird mit modularen Komponenten aufgebaut. Dem App-Kontext werden zwei voneinander getrennte Root-Module zur Verfügung gestellt. Ein Modul besteht aus [1..N] Page-Modulen. Diese Page-Module wiederum setzen sich aus eigens erstellten Komponenten und vordefinierten Komponenten des Frameworks zusammen. Das Verhalten dieser Komponenten wird durch deren Scripts und allgemein verfügbaren Services definiert. Der Mobile-Client wird so nach einem Objekt-orientierten Paradigma aufgebaut.

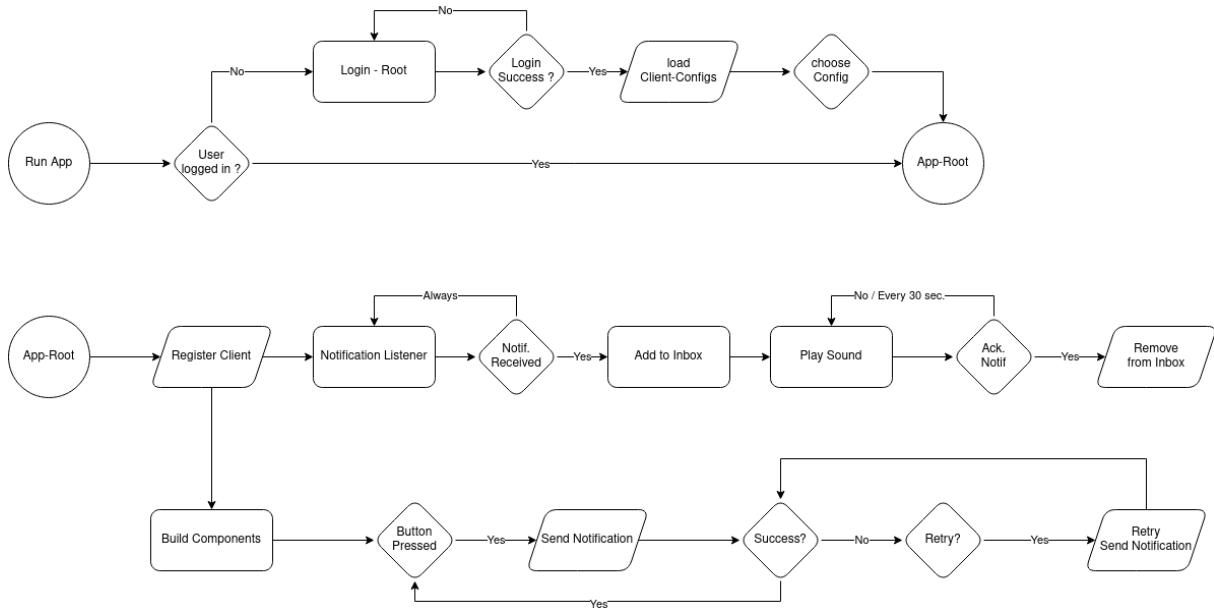


Abbildung 5.4: Mobile-Client Flow Chart

Das eigentliche *User Interface* besteht aus dem Homescreen, der es dem Benutzer erlaubt Nachrichten zu versenden, und der Inbox welche eingegangene Nachrichten anzeigen. Diese Ansicht ist erst nach erfolgreicher Authentifizierung erreichbar. Hat sich der Benutzer erfolgreich angemeldet, erhält er eine Auswahl der ihm zur Verfügung stehenden Konfigurationen. Mit der Auswahl einer dieser Konfigurationen werden die vordefinierten Notification Buttons geladen und auf dem Homescreen erstellt. Innerhalb des App-Root Kontextes sind zwei Workflows parallel aktiv. Zum einen können die vordefinierten Nachrichten durch Betätigen des entsprechenden Buttons versendet werden. Die Empfänger werden durch die Rule-Engine des Cloud Service ermittelt. Bei einem Fehlschlag wird dies dem Benutzer via Pop-Up mitgeteilt und er kann entscheiden, ob die Nachricht nochmals neu versendet werden soll. Gleichzeitig ist immer der Listener aktiv, der auf Benachrichtigungen des Messaging Services wartet. Dieser wird auf dem Client eingegangene Nachrichten in der Inbox ablegen und ein akustisches Signal abspielen. Wird die Meldung nicht innerhalb eines definierten Zeitraums quittiert, wird das akustische Signal wiederholt.

5.2.4 User Interface

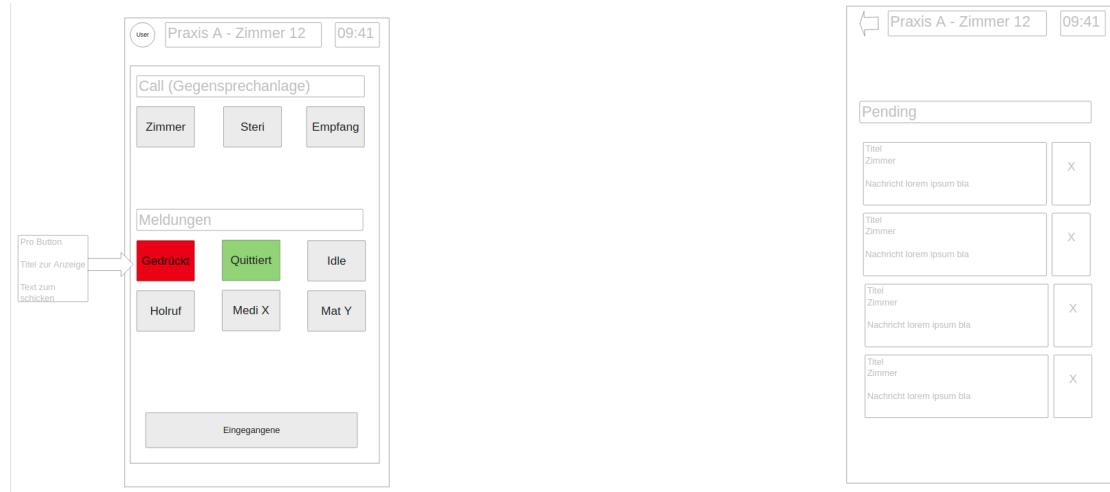


Abbildung 5.5: HomeScreen Mockup

Abbildung 5.6: Inbox Mockup

Die Buttons der Meldungen werden in einem 3x3 Grid auf dem Homescreen dynamisch angelegt. Nach Betätigung eines Buttons wird dieser in Rot eingefärbt, bis eine Rückmeldung über Erfolg oder Misserfolg vom Cloud Service eingegangen ist. Die gezeigten Mockups wurden zusammen mit dem Kunden entwickelt. Sie zeigen dementsprechend bereits Bereiche zur Bedienung einer Gegensprechanlage. Hier sollten die Gesprächspartner direkt mit einem entsprechenden Button angewählt werden. Diese Funktionalität ist allerdings ausserhalb des Projektumfangs gefallen und wird deshalb nicht weiter beschrieben⁴.

Die Inbox besteht aus einer ScrollView, welche eingegangene Nachrichten als Cards darstellt. Eine Nachricht enthält:

- Titel der Nachricht
- Absender der Nachricht
- Detail Text
- Icon

⁴Siehe Kapitel 3

5.3 Cloud Service

5.3.1 Architektur

Der Cloud Service 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.

Durch die klare Trennung der Verantwortungsbereiche der beiden Domänen, wäre es möglich diese in einzelne Microservices aufzuteilen. Dies würde das Skalieren der Applikationen vereinfachen. Die Aufteilung auf mehrere Microservices macht das System als ganzes aber komplexer. Insbesondere Betrieb und Entwicklung wird aufwändiger, da diese Aufwände nun für mehrere Applikationen anfallen. Für den Umfang dieser Arbeit wird deshalb darauf verzichtet, den Cloud Service als echtes Micro Service System umzusetzen. Der Cloud Service wird als eine einzelne Spring Boot Applikation umgesetzt. Innerhalb dieser Applikation sollen die Domänen Configuration und Notification allerdings, wo immer mit vertretbarem Aufwand möglich, getrennt bleiben. Direkte Abhängigkeiten zwischen den beiden Domänen sind zu vermeiden. An den Stellen wo Informationen aus der anderen Domäne nötig sind, sollen diese über ein Rest Interface abgefragt werden. So kann die Applikation in Zukunft einfach und auf die tatsächlichen Bedürfnisse zugeschnitten in mehrere Microservices aufgeteilt werden.

Um einfache Erweiterbarkeit zu gewährleisten, wird der Cloud Service nach dem Prinzip von Onion Architecture aufgebaut.

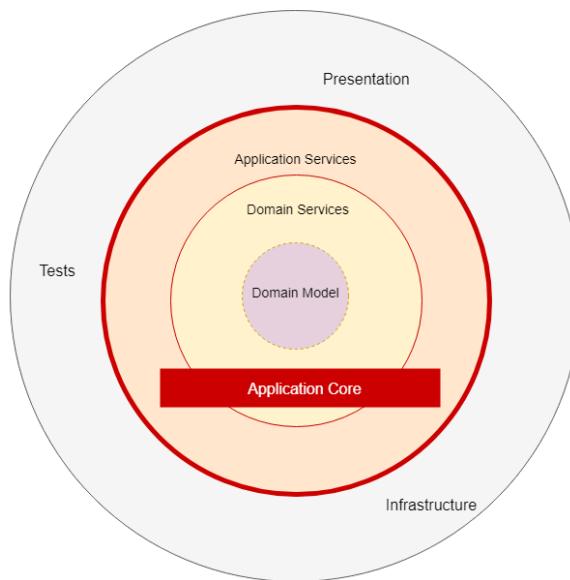


Abbildung 5.7: Onion Architecture

In Onion Architecture wird die Applikation in Layer aufgeteilt.[20] Im Zentrum des Modells steht die Domäne selbst. Sie wird durch den Domain Layer abgebildet. Der Domain Layer darf nur Abhängigkeiten auf sich selbst haben. Umgekehrt dürfen aber alle anderen Layers Abhängigkeiten auf den Domain Layer haben. Die nächste Schicht im Modell ist der Domain Service Layer. Dieser bietet die fachliche Logik und definiert die Verhaltensweise des Domain Layers. Der Layer Application Services bildet die Brücke zwischen externer Infrastruktur und Domain Services. Dies beinhaltet Repository Services für die Schnittstelle zu persistentem Speicher und Rest Controllers für Schnittstellen zu anderen Applikatio-

nen. In der äussersten Schicht steht der Infrastructure Layer. Dieser beinhaltet externe Systeme, welche von der Applikation benötigt werden, wie Datenbanken, Messaging Services und Applikationen die auf Web APIs welche der Applikation zugreifen.

Um zu garantieren, dass keine ungewollten Abhängigkeiten zwischen Layern bestehen, können die Layers in eigene Module verpackt werden. Dies erhöht jedoch die interne Komplexität der Applikation. Das Onion Modell wird innerhalb des Cloud Services deshalb nicht durch Module, sondern durch eine vorgegebene Packagestruktur umgesetzt.

```
ch.fhnw.ip5.praxiscloudservice
├── api
├── config
├── Domäne
├── persistence
├── service
└── web
```

Abbildung 5.8: Package Struktur Cloud Service

Im Zentrum steht das Package Domäne. Es beinhaltet alle Domänenobjekte und stellt alleine den Domäne Layer dar. Der Domain Service Layer wird durch das Package Service abgebildet. Hier werden sämtliche Domain Services implementiert. Die Packages persistence und web beinhalten schliesslich den Application Service Layer. Dabei definiert das Package persistence Services welche für Interaktion mit der Datenbank verwendet werden. Das Package web definiert die REST-Endpunkte und Clients, welche für die Kommunikation zwischen den Domänen und anderen Services benötigt werden. Die Objekte, welche darüber ausgetauscht werden, sind im Package api definiert. So werden direkte Abhängigkeiten zwischen den Domänen Notification und Configuration vermieden. Das Package api beinhaltet zudem Interfaces für Domain Services und Exceptions. Diese werden im Package services implementiert und im package Web verwendet. Letztlich beinhaltet das config die technische Konfiguration der Applikation.

5.3.2 Domäne Configuration

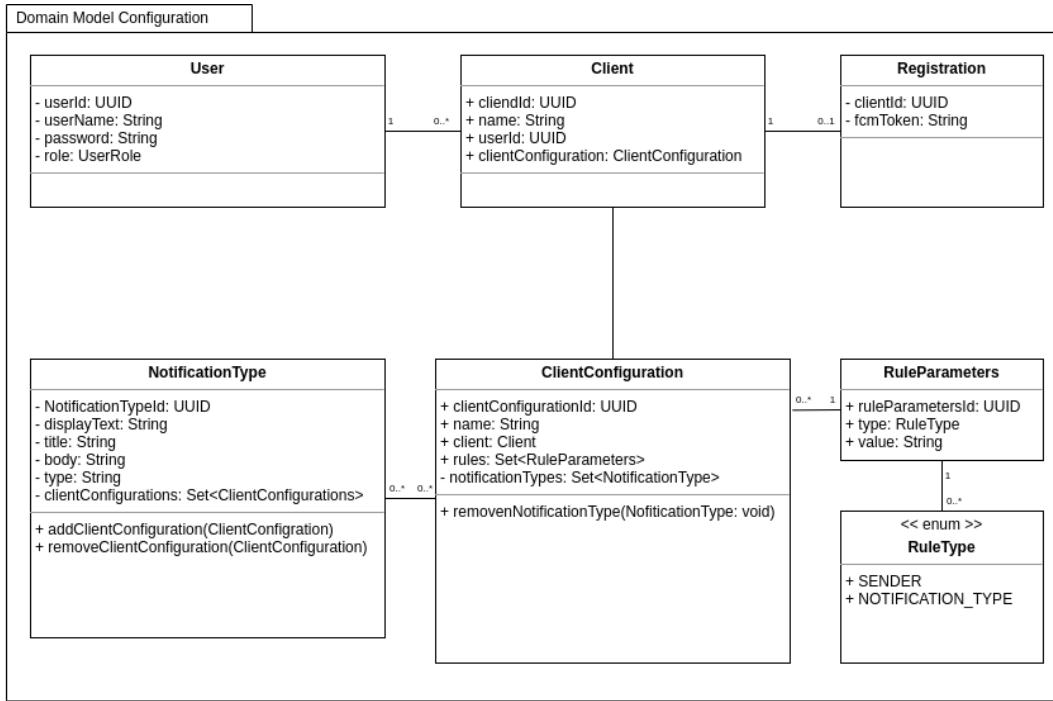


Abbildung 5.9: Domänenmodell Configuration

Das Herzstück der Configuration Domäne ist die ClientConfiguration. Die ClientConfiguration beinhaltet die Informationen, welche Benachrichtigung ein Endgerät versenden kann und welche Benachrichtigungen es empfangen soll. Ein Endgerät wird dabei durch die Entität Client dargestellt. Client und ClientConfiguration sind getrennt, damit Clients auch ohne Configuration erfasst werden können. So kann ein Administrator bereits Geräte im System erfassen, ohne Sie vollständig zu konfigurieren. Die konkrete Konfiguration von Benachrichtigungen erfolgt über die Entitäten NotificationType und RuleParameters. Ein NotificationType ist die Grundlage für eine Benachrichtigung. Er definiert den Inhalt, der mit einer Benachrichtigung versendet wird. Die Liste von NotificationTypes auf einer ClientConfiguration definiert, welche Benachrichtigungen der zugehörige Client versenden kann. Welche Benachrichtigungen ein Client empfängt, wird über die Liste von RuleParameters auf der ClientConfiguration definiert. Die konfigurierten RuleParameters werden von der RulesEngine ausgewertet, um relevante Empfänger für eine Benachrichtigung zu finden. Dabei kann die Entität RuleParameters für alle Regeln wiederverwendet werden. Die RuleEngine kann anhand des RuleType bestimmen, wie RuleParameters ausgewertet werden.

Zur Verwaltung von Konfigurationen wird weiter die Entität PraxisUser definiert. Sie stellt einen Benutzer des Praxisrufsystems dar. Jeder Benutzer kann die Rollen Admin oder User haben. Ein Benutzer mit der Rolle Admin hat Recht die Konfiguration über das Admin UI zu verwalten. Ein Benutzer mit der Rolle User ist berechtigt, den Mobile Client zu verwenden.

Für die Anbindung an den Messaging Service ist zudem die Entität Registration notwendig. Die Registrierung eines Clients beinhaltet die technische Identifikation, mit der ein Client sich beim Messaging Service registriert hat. Ein Client kann immer nur genau eine Registration haben.

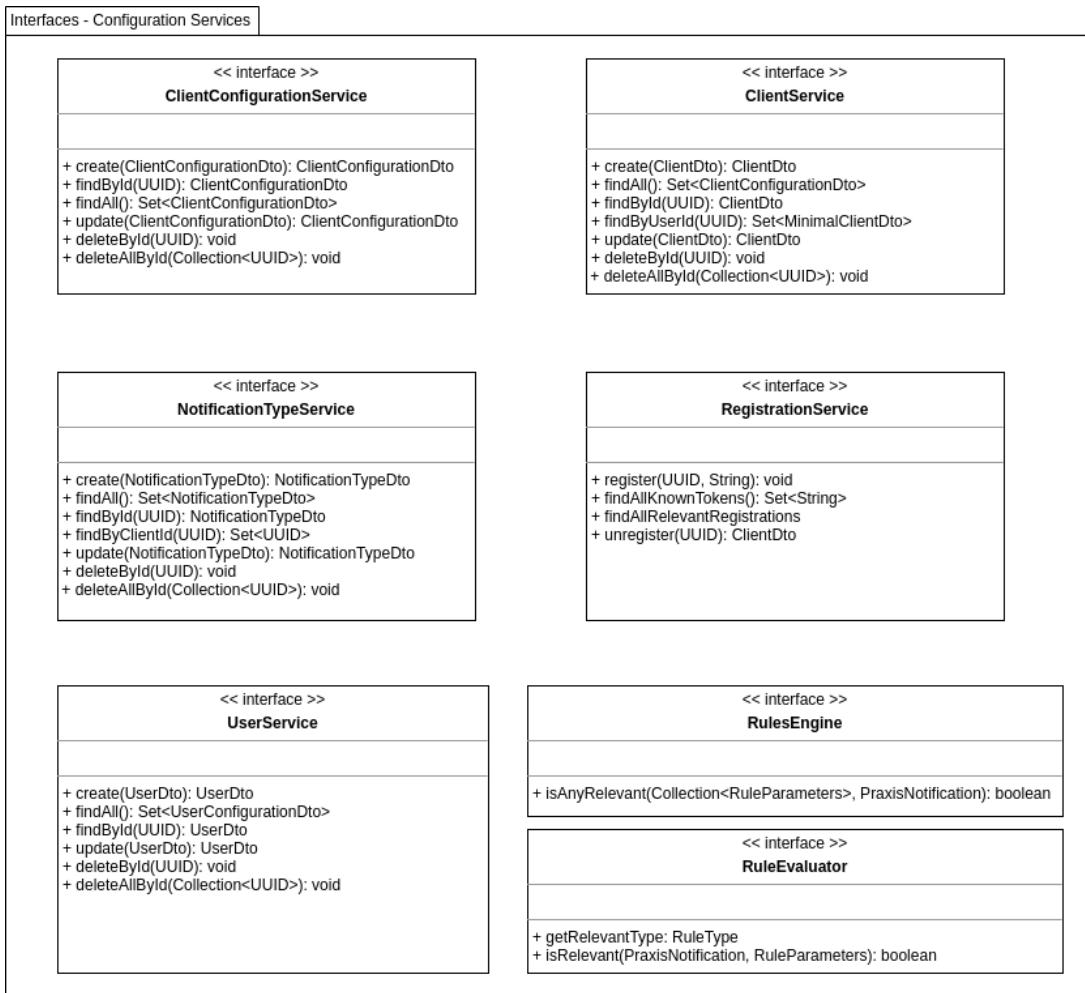


Abbildung 5.10: Klassendiagramm Configuration Service Interfaces

Die Configuration Domäne beinhaltet Services zum Lesen, Erstellen und Löschen der Domänenobjekte. Diese Verwaltungsfunktionen werden über eine REST-API exponiert.⁵ Der RegistrationService bietet dabei eine Ausnahme. Er bietet die Funktionen, Registrierungen zu erfassen und aktualisieren. Diese werden verwendet, wenn sich ein Mobile Client beim Cloud Service registriert. Er bietet zudem die Möglichkeit die Identifikation (Token) aller relevanten Empfänger für eine gegebene Benachrichtigung zu finden. Diese Auswertungen werden mithilfe der RulesEngine gemacht. Die RulesEngine bewertet die erfassten RulesParameter mithilfe der RuleEvaluators und findet alle relevanten Einträge.

Die RulesEngine ermöglicht es, dem Cloud Service festzustellen, welche Benachrichtigungen an welche Clients versendet werden sollen. Dazu bietet sie eine einzelne öffentliche Methode. Diese nimmt eine Liste von Regelparametern (RuleParameters) und eine Notification entgegen. Zurückgegeben wird ein Boolean Wert, der sagt, ob mindestens eine der übergebenen Regelparameter für die Benachrichtigung relevant ist. Da jeder aktive Client eine ClientConfiguration definiert, die eine Liste an Regelparametern beinhaltet kann so mit der RulesEngine bewertet werden, ob ein bestimmter Client sich für eine Benachrichtigung interessiert.

Die Auswertung der einzelnen Regelparameter innerhalb der RulesEngine wird an einzelne RuleEvaluator delegiert. Umgesetzt wird dieses Konzept mit einem Strategy Pattern.[21] RuleEvaluator definiert das Strategy Interface. Der Zugriff auf die einzelnen Strategies innerhalb der RulesEngine wird über eine RuleEvaluatorFactory gelöst. Diese Factory kennt alle existierenden RuleEvaluator Instanzen und bie-

⁵Siehe Kapitel 5.3.4

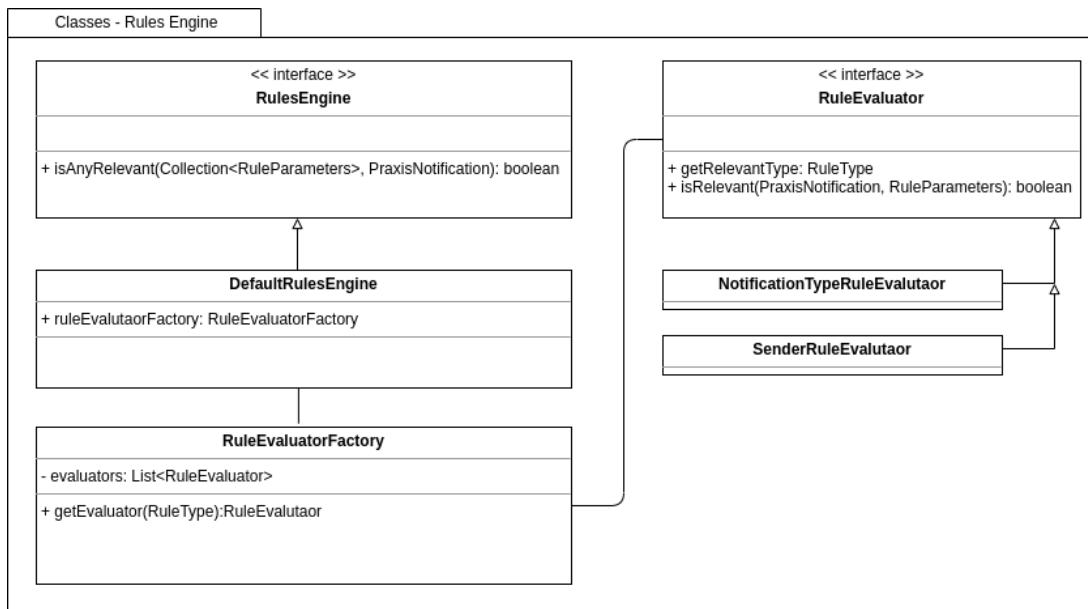


Abbildung 5.11: Klassendiagramm Rules Engine

tet eine öffentliche Methode, über welche der RuleEvaluator für einen bestimmten Typ geladen werden kann. Über die Dependency Injection des Spring Frameworks, kann diese RuleEvaluatorFactory einfach und erweiterbar implementiert werden:

```

1  @Service
2  public class RuleEvaluatorFactory {
3
4      private final Map<RuleType, RuleEvaluator> evaluators = new HashMap<>();
5
6      @Autowired
7      public RuleEvaluatorFactory(List<RuleEvaluator> evaluatorInstances) {
8          evaluatorInstances.forEach(e -> evaluators.put(e.getRelevantType(), e));
9      }
10
11     public RuleEvaluator get(RuleType ruleType) {
12         return evaluators.get(ruleType);
13     }
14 }
  
```

Listing 4: RuleEvaluatorFactory.java

Über den `@Autowired` Konstruktor nimmt die Factory eine Liste des Types RuleEvaluator entgegen. Die Spring Dependency Injection wird hier automatisch alle verfügbaren RuleEvaluator Instanzen in einer Liste sammeln und als Konstruktorparameter entgegennehmen. Da jeder RuleEvaluator eine Methode bietet, über die der relevante RuleType abgefragt werden kann, kann nun aus dieser Liste eine Map gebaut werden, die RuleTypes auf die relevanten RuleEvaluator Instanzen mapped. Das Laden eines RuleEvaluators anhand des RuleTypes kann nun über ein einfaches Lookup in der Map erfolgen. Werden in der Zukunft weitere RuleTypes unterstützt, reicht es den entsprechenden RuleEvaluator zu implementieren. Der neue RuleEvaluator wird danach automatisch über die RuleEvaluatorFactory verfügbar sein. Anpassungen an der RulesEngine oder der RuleEvaluatorFactory sind nicht nötig.

5.3.3 Domäne Notification

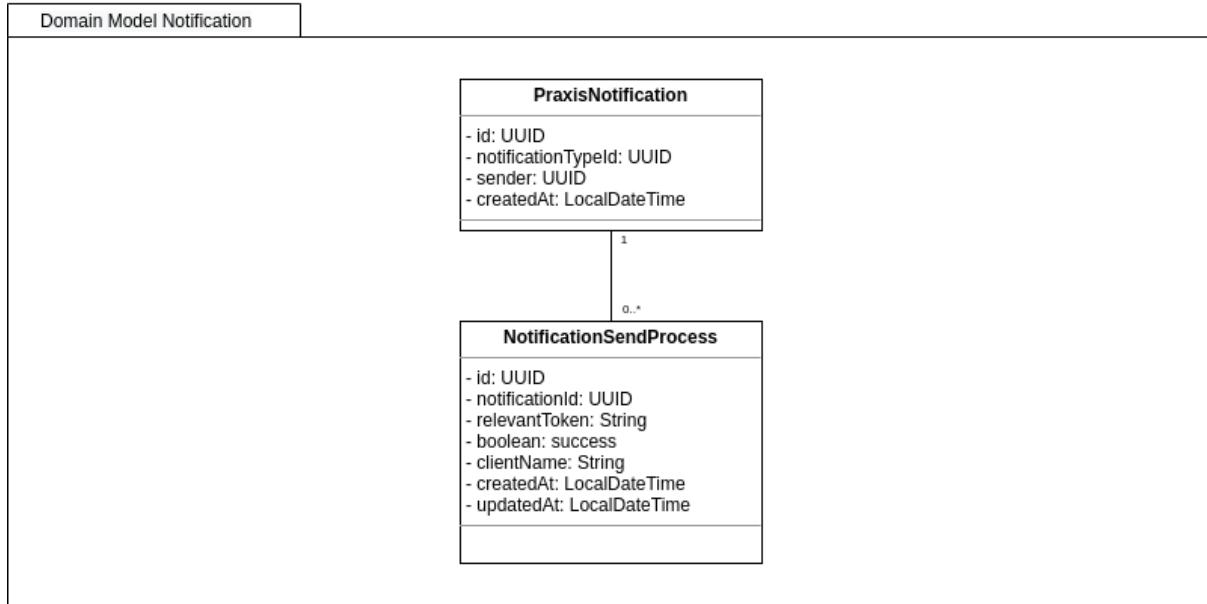


Abbildung 5.12: Domänenmodell Notification

In der Domäne Notification werden zwei Entitäten definiert. Die Entität PraxisNotification stellt eine Benachrichtigung dar, die über den Cloud Service versendet wird. Jede Notification die der CloudService erhält wird bereits vor dem Versenden persistiert. Nach dem eine Benachrichtigung als PraxisNotificationpersistiert wurde, wird sie an die Empfänger versendet. Dabei wird für jeden Empfänger ein SendNotificationProcess erstellt. Dieser SendNotificationProcess beinhaltet eine Referenz auf die versendete Benachrichtigung und den Status des Sendeprozesses. Dies dient zum einen der Nachvollziehbarkeit. Zum anderen ermöglicht es, fehlgeschlagene Benachrichtigungen nur für die Empfänger zu wiederholen, für die der Versand fehlgeschlagen ist.

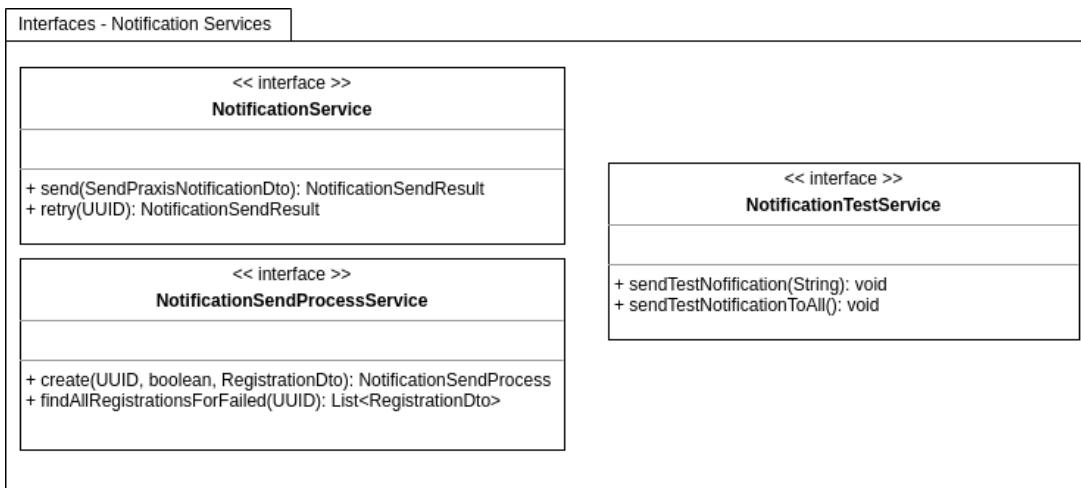


Abbildung 5.13: Klassendiagramm Notification Service Interfaces

In der Notification Domäne werden drei Domain Services definiert. Der NotificationService bietet Methoden, um eine Notifikation zu versenden und zu wiederholen. Für das initiale Versenden werden die Informationen benötigt, um eine Notifikation zu erstellen und die Empfänger zu identifizieren. Im Fall der Wiederholung werden diese Informationen nicht mehr benötigt. Die Notifikation und der Status des

Versandprozesses wurden beim ersten Versuch sie zu versenden bereits persistiert. Als zweiter Service definiert der NotificationSendProcessService dazu Sendevorgänge zu erstellen und fehlgeschlagene Sendevorgänge zu finden. Der dritte Service in der Notification Domäne dient nur zu Test- und Administrationszwecken. Der NotificationTestService kann verwendet werden, um einem einzelnen Client eine Testnachricht zu senden oder allen registrierten Clients eine Nachricht zu schicken. Als Benachrichtigung wird dabei immer eine vom Cloud Service definierte Benachrichtigung versendet, welche nur Platzhalterwerte beinhaltet.

5.3.4 Laufzeitmodell

Im Folgenden werden die Abläufe für die Registrierung von Mobile Clients sowie das Versenden und Empfangen von Benachrichtigungen im Detail definiert.

Client Registration

Damit ein Client Benachrichtigungen empfangen und versenden kann, muss er sich bei Messaging Service und Cloud Service registrieren. Hier wird dieser Ablauf spezifiziert. Als Vorbedingung wird dabei angenommen, dass mindestens ein Client inklusive ClientConfiguration erfasst und dem Praxismitarbeiter zugewiesen wurde.

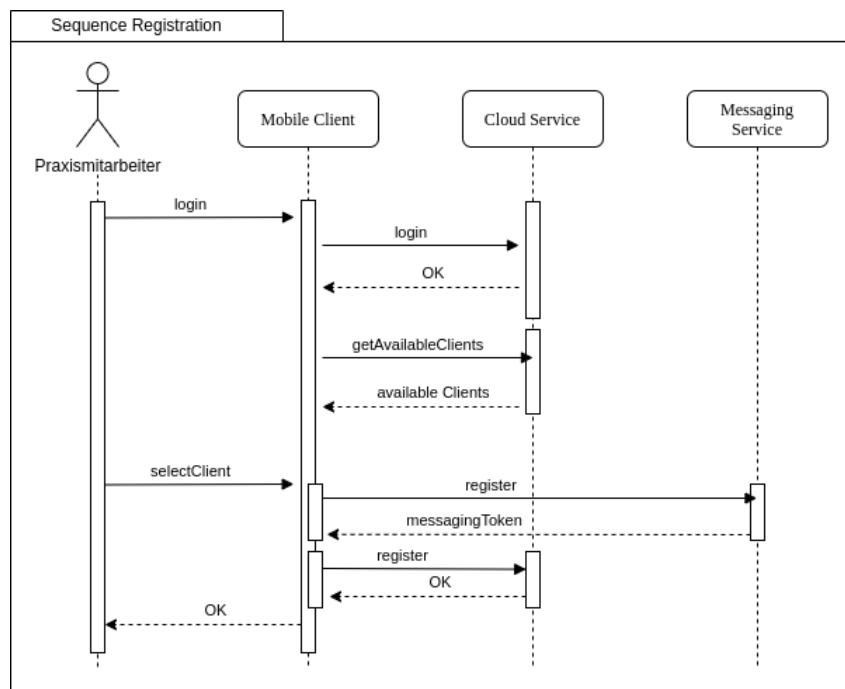


Abbildung 5.14: Ablauf Registration

In einem ersten Schritt muss sich der Praxismitarbeiter im Mobile Client anmelden. Die Anmeldung wird im Cloud Service ausgewertet. Hat er gültige Benutzerdaten angegeben, werden Informationen zu allen verfügbaren Konfigurationen vom Cloud Service geladen und der Benutzer kann die gewünschte Konfiguration auswählen. Dabei werden nur Name und Id der Konfigurationen geladen, damit nicht mehr Daten als nötig übertragen werden. Wählt Benutzer die gewünschte Konfiguration, werden alle dafür konfigurierten NotificationTypes geladen und im UI die entsprechenden Buttons erstellt. Direkt nach dem Laden der Konfiguration registriert sich der Mobile Client beim Messaging Service. Als Antwort erhält er ein eindeutiges Token, welches verwendet werden kann, um an diesem Client Nachrichten zu senden. Der Mobile Client registriert sich darauf mit dem erhaltenen Token und der ausgewählten

Konfiguration beim Cloud Service. Nach diesem Ablauf ist der Client bei Messaging Service und Cloud Service registriert und bereit Benachrichtigungen zu empfangen.

Benachrichtigung versenden und empfangen

Die zentrale Funktion des Praxisrufsystems ist das Versenden und Empfangen von Benachrichtigungen. Hier wird dieser Ablauf im Erfolgsfall beschrieben. Als Vorbedingung wird angenommen, dass Konfiguration und Registrierung gemäss Abbildung 5.14 für zwei Clients abgeschlossen ist. Dabei ist einer der Clients konfiguriert, Benachrichtigungen vom anderen Client zu empfangen.

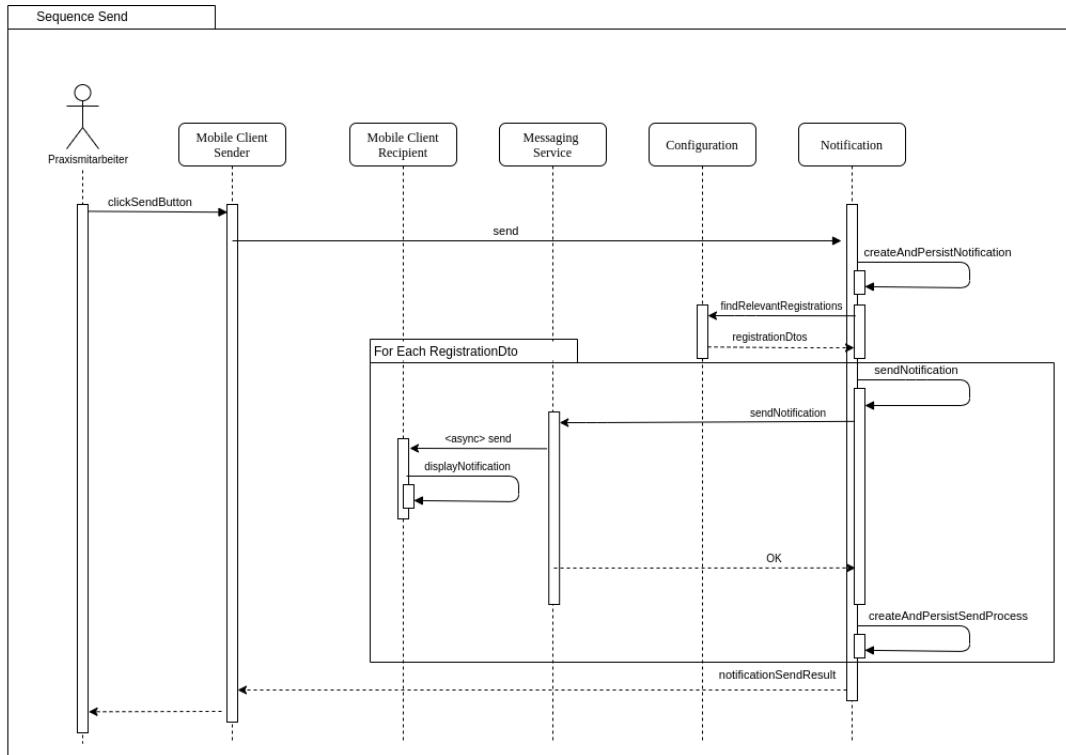


Abbildung 5.15: Ablauf Benachrichtigung Senden und Empfangen

Nachdem der Benutzer das Versenden einer Benachrichtigung auslöst, wird eine Anfrage an den NotificationController im Cloud Service gesendet. Darin enthalten sind die Id des Senders sowie die Id des NotificationTypes. Der NotificationController macht in der Folge eine Anfrage an den ConfigurationController, um alle relevanten Empfänger zu finden. Im ConfigurationController werden die erfassten RuleParameters aller konfigurierten ClientConfigurations ausgewertet. Darauf wird eine Liste der Registrierungen aller relevanten Empfänger zurückgegeben. Sobald die Empfänger geladen sind, erstellt der NotificationController eine Benachrichtigung aus dem gegebenen NotificationType. Diese Benachrichtigung wird als PraxisNotification persistiert und an den MessagingService übergeben. Der Messaging Service stellt die Zustellung anhand der mitgegebenen Registrierung sicher. Nach der Übermittlung an den Messaging Service wird im CloudService pro Empfänger ein NotificationSendProzess erstellt der den Status für diesen Empfänger dokumentiert. Der NotificationController sendet darauf eine Bestätigung an den Mobile Client. Diese Antwort beinhaltet die technische Id der versendeten Benachrichtigung und das Resultat, ob die Benachrichtigung an alle Empfänger versendet werden konnte. Auf Empfängerseite wird die Benachrichtigung verarbeitet und angezeigt, sobald die Zustellung durch den Messaging Service erfolgt ist. Diese Zustellung erfolgt asynchron. Dementsprechend kann der Sender nur informiert werden, wenn die Zustellung an den Messaging Service fehlgeschlagen ist.

Benachrichtigung wiederholen

Die zentrale Funktion des Praxisrufsystems ist das Versenden und Empfangen von Benachrichtigungen. Hier wird dieser Ablauf im Fehlerfall beschrieben. Als Vorbedingung wird angenommen, dass Konfiguration und Registrierung gemäss Abbildung 5.14 für zwei Clients abgeschlossen ist. Dabei ist einer der Clients konfiguriert, Benachrichtigungen vom anderen Client zu empfangen. Es wurde eine Benachrichtigung gemäss Abbildung 5.15 versendet, das Versenden an mindestens einen Client ist fehlgeschlagen und der Cloud Service hat ein negatives SendNotificationResult zurückgegeben.

Nach dem Erhalt des negativen Resultats zeigt der Mobile Client einen Dialog an. Dieser Dialog informiert den Benutzer über den Fehler und fragt an, ob die fehlgeschlagenen Benachrichtigungen wiederholt werden sollen. Bestätigt der Benutzer wird diese Anfrage, wird eine Anfrage an den NotificationController gesendet um das Versenden zu wiederholen. Diese Anfrage beinhaltet die technische Id der fehlgeschlagenen Benachrichtigung. Der NotificationController findet alle negativen Einträge für diese Benachrichtigung in der NotificationSendProcess Tabelle. Anschliessend wird jeder gefundene SendProcess anhand der Tokens in dieser NotificationSendProcess Instanzen wiederholt. Das Versenden, Empfangen und die Rückmeldung an den Mobile Client erfolgen analog zum Ablauf "Benachrichtigung versenden und empfangen".

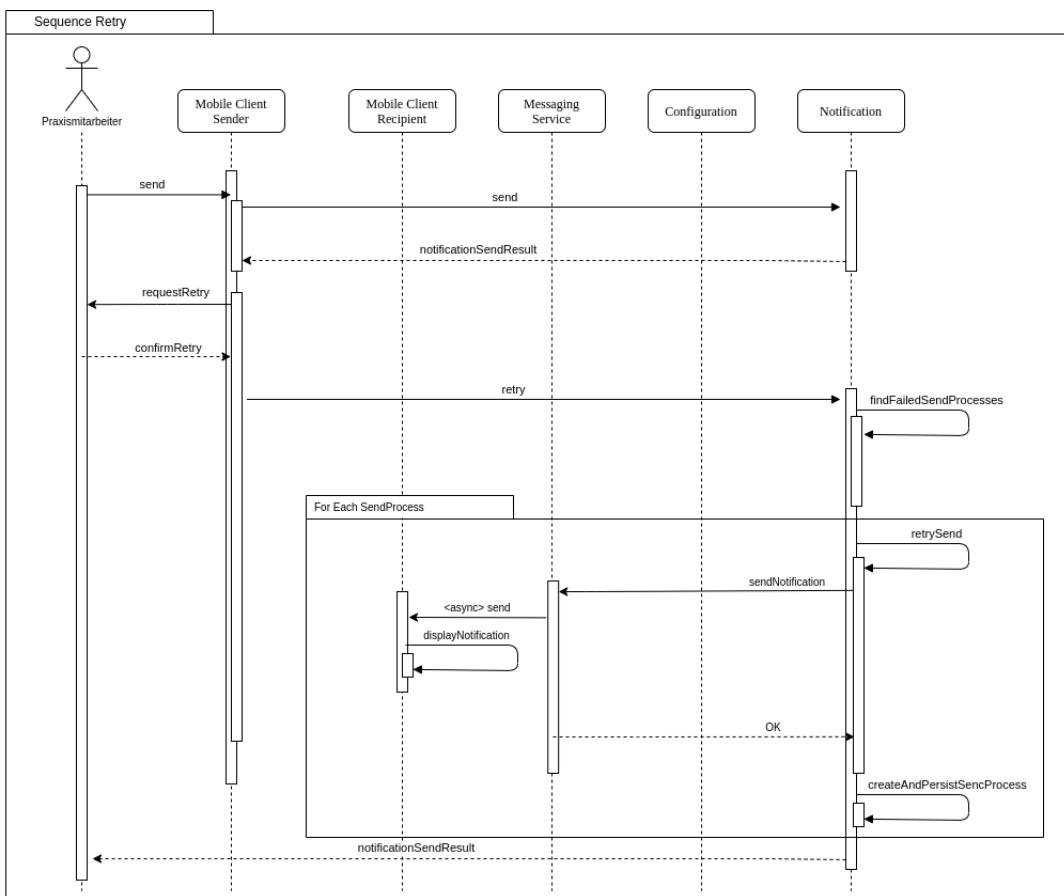


Abbildung 5.16: Ablauf Benachrichtigung Wiederholen

5.3.5 API

Um die Verwaltung der Konfigurationen zu ermöglichen, bietet der Cloud Service eine REST-API an. Der Cloud Service bietet Verwaltungs-APIs für die folgenden Domänenobjekte:

Domänenobjekt	Entity Name
Client	clients
Client Configuration	client-configurations
Notification Types	notification-types
Users	users

Für jedes dieser Domänenobjekte ist ein dedizierter Endpunkt definiert, der unter dem Pfad /api/entity-name erreichbar ist. Jeder dieser Kontroller bietet eine API zu Verwaltung dieses Domänenobjekts, welche dem folgenden Schema folgt:

Action	HTTP	Pfad	Body	Response
Alle Elemente lesen	GET	/api/entity-name	-	[EntityDto]
Einzelnes Element lesen	GET	/api/entity-name/id	-	EntityDto
Neues Element erstellen	POST	/api/entity-name	EntityDto	EntityDto
Bestehendes Element ändern	PUT	/api/entity-name	EntityDto	EntityDto
Einzelnes Element löschen	DELETE	/api/entity-name/id	-	-
Mehrere Elemente löschen	DELETE	/api/entity-name/ids	-	-

Eine Ausnahme bildet hier das Domänenobjekt Registration. Hier wird analog zum Domänenservice für Registrations⁶ folgende API angeboten:

Aktion	HTTP	Pfad	Body	Response
Registrierung aktualisieren	POST	/api/registrations	ClientId, Token	-
Registrierung entfernen	DELETE	/api/registrations	ClientId	-
Relevante Registrierung finden	POST	/api/registrations/tokens	Notification-Dto	[Registration-Dto]

Benachrichtigungen

Zum Versenden von Benachrichtigungen bietet der Cloud Service folgende Endpunkte an:

Aktion	HTTP	Pfad	Body	Response
Nachricht versenden	POST	/api/notifications/send	NotificationDto	Notification-Response-Dto
Fehlgeschlagene erneut versenden	POST	/api/notifications/retry	NotificationId	Notification-Response-Dto

⁶Siehe Kapitel 5.3.2

5.3.6 Authentifizierung

Die Authentifizierung von Benutzern wird mittels Spring-Security gelöst. Der Einfachheit halber werden die User in einer eigenen Datenbanktabelle verwaltet. Mit einem vertretbaren Mehraufwand könnte diese Verantwortlichkeit jedoch auch noch an einen externen Provider wie z.B. Keycloak delegiert werden.

Der User authentifiziert sich mit seinem Username und Passwort (BasicAuth) erstmalig über den Login-Endpoint. Der Cloud Service stellt mit dem DefaultUserService eine eigene Implementation für die von Spring-Security verwendeten Komponenten UserDetailsService und AuthenticationProvider bereit (siehe Abbildung 5.18). Hier werden die übergebenen Credentials überprüft und bei erfolgreicher Anmeldung ein Authentication Objekt mit den entsprechenden Benutzerrollen im Security-Context abgelegt.[22]

Aus diesem Objekt wird ein JWT-Token generiert, welches 24h lang gültig ist und dem Client zurückgegeben. Für alle weiteren Request während der laufenden Session authentifiziert sich der Client mit diesem JWT-Token (BearerToken Auth).

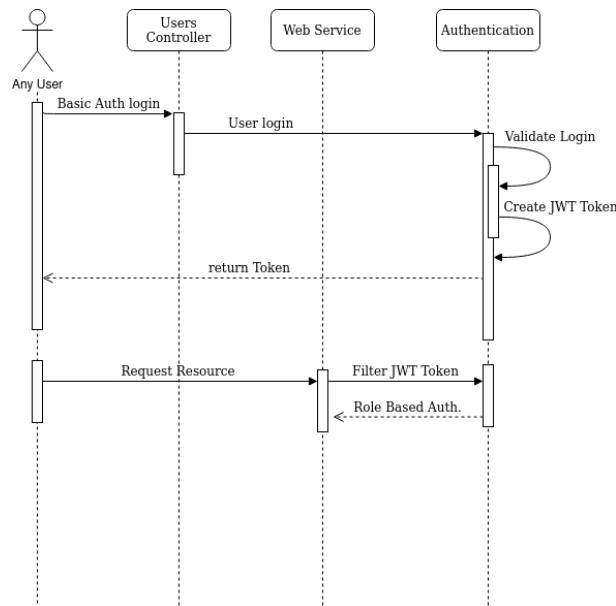


Abbildung 5.17: Authentifizierung-Sequenz

Der Vorteil an JWT ist dabei, dass zum einen keine SessionId als Cookie im Browser gespeichert wird.[23] Und zum Anderen, dass innerhalb des JWT Tokens zusätzliche Metadaten gespeichert werden können.[24] Wir verwenden dies um die Rolle sowie die Id des Benutzers bei den Requests mitliefern zu können.

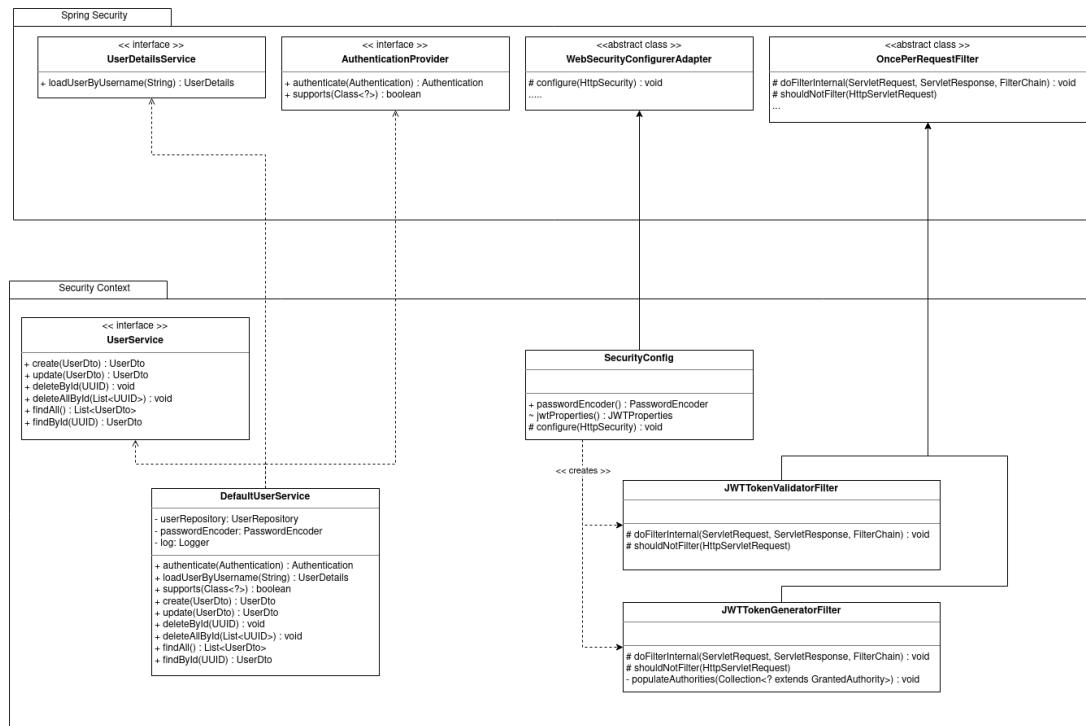


Abbildung 5.18: SecurityContext-Klassendiagramm

5.3.7 Autorisierung

Aktuell werden die Rollen User und Administrator verwendet. Das Rollensystem lässt sich ohne grösseren Aufwand beliebig im entsprechenden Enum erweitern. Die Abfrage der Berechtigung wird für das gesamte System zentral in der **SecurityConfig** (Siehe Abbildung 5.18) geprüft. Dabei werden die entsprechenden Rollen für den jeweiligen HTTP-Request abgefragt.

5.4 Admin UI

5.4.1 Framework Grundlagen

Auf Wunsch des Kunden wird das Admin UI als Web-Applikation auf Basis des React Admin Frameworks entwickelt. Bei React-Admin baut auf dem React-Framework auf und vereinfacht das Erstellen, Lesen, Bearbeiten und Löschen von Datensätzen.[25] Es setzt auf den folgenden Technologien auf:

- React
- material UI
- React Router
- Redux
- Redux Saga
- React Final Form

Das Framework abstrahiert eine grosse Bandbreite an Funktionalität welche häufig in Administrations-Oberflächen gefordert sind. Die Voraussetzung hierfür ist eine konsistente API welche für alle Routen dieselben Endpunkte anbieten [25].

5.4.2 Anwendung

Die konfigurierbaren Elemente werden jeweils in einer eigenen Komponente verwaltet.

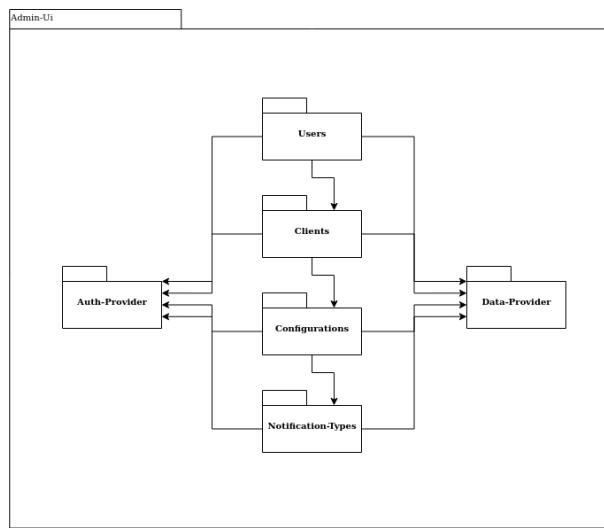


Abbildung 5.19: Admin-Ui Package Diagramm

Die Anfragen werden von jeweils von einem Data-Provider durchgeführt, der an die API des Cloudservices angepasst wird. Vor jedem Request wird der Auth-Provider aufgerufen um zu verifizieren, dass der aktuelle Benutzer auch berechtigt ist, die gewünschte Aktion durchzuführen.

5.5 Proof Of Concept

Um sicherzustellen, dass die Anforderungen an das System mit den gewählten Technologien umgesetzt werden können, wird zunächst ein Proof Of Concept implementiert. Dieser Proof Of Concept hat einen deutlich kleineren Funktionsumfang als das Endprodukt. Im Wesentlichen muss der Proof Of Concept beweisen, dass es möglich ist mit den gewählten Technologien Benachrichtigungen zu Versenden und zu empfangen.

5.5.1 Funktionale Anforderungen

Um dies zu ermöglichen werden die folgenden Features in eingeschränktem Umfang umgesetzt.

F01 - Benachrichtigungen Versenden

Mit dem Proof Of Concept muss es möglich sein, Benachrichtigungen von einem Client zu versenden. Dementsprechend wird das Szenario “Benachrichtigung versenden” mit den folgenden Einschränkungen umgesetzt:

- Es erfolgt kein Login und keine Authentifizierung.
- Es gibt nur einen vordefinierten Button.
- Es wird immer dieselbe vordefinierte Benachrichtigung versendet.
- Es werden keine Empfänger konfiguriert, die Benachrichtigung wird zurück an den Absender versendet.

F02 - Benachrichtigungen empfangen

Mit dem Proof Of Concept muss es möglich sein, Benachrichtigungen mit einem Client zu empfangen. Dementsprechend wird das Szenario “Benachrichtigung empfangen” mit den folgenden Einschränkungen umgesetzt:

- Es wird keine Liste von Benachrichtigungen geführt.
- Die Benachrichtigung wird in einem einfachen Textfeld im Mobile Client angezeigt.

F04 - Über Benachrichtigungen Notifizieren

Mit dem Proof Of Concept muss es möglich sein, über den Einfang von Benachrichtigungen notifiziert zu werden. Dementsprechend werden die Szenarien “Foreground” und “Background” mit den folgenden Einschränkungen umgesetzt:

- Die Notifizierung erfolgt ohne Audio Signal.

5.5.2 Technische Anforderungen

Damit der Proof Of Concept aussagekräftig ist, müssen die folgenden technischen Anforderungen umgesetzt werden:

T01 - IPad Client

Der für den Proof Of Concept umgesetzte Client muss auf einem IPad funktionieren und alle Anforderungen die an den Proof Of Concept gestellt werden erfüllen. Dazu muss er Anfragen an den Cloud Service senden und Benachrichtigungen vom Messaging Service empfangen können.

T04 - AWS Platform

Der Cloud Service muss auf AWS deployt werden. Dabei muss die Plattform so konfiguriert werden, dass Anfragen vom Mobile Client empfangen und Benachrichtigungen an den Messaging Service gesendet werden können.

5.5.3 Laufzeitsicht

Im Wesentlichen muss der Proof Of Concept beweisen, dass es möglich ist mit den gewählten Technologien Benachrichtigungen zu Versenden und zu Empfangen.

Der Benutzer muss den Mobile Client auf dem IPad öffnen und über einen Button eine Benachrichtigung versenden können. Diese Benachrichtigung wird an den Cloud Service übermittelt. Dieser stellt die Benachrichtigung über den Messaging Service einem Empfänger zu. Im Rahmen des Proof Of Concept wird eine Benachrichtigung immer an den Sender zurückgesendet.

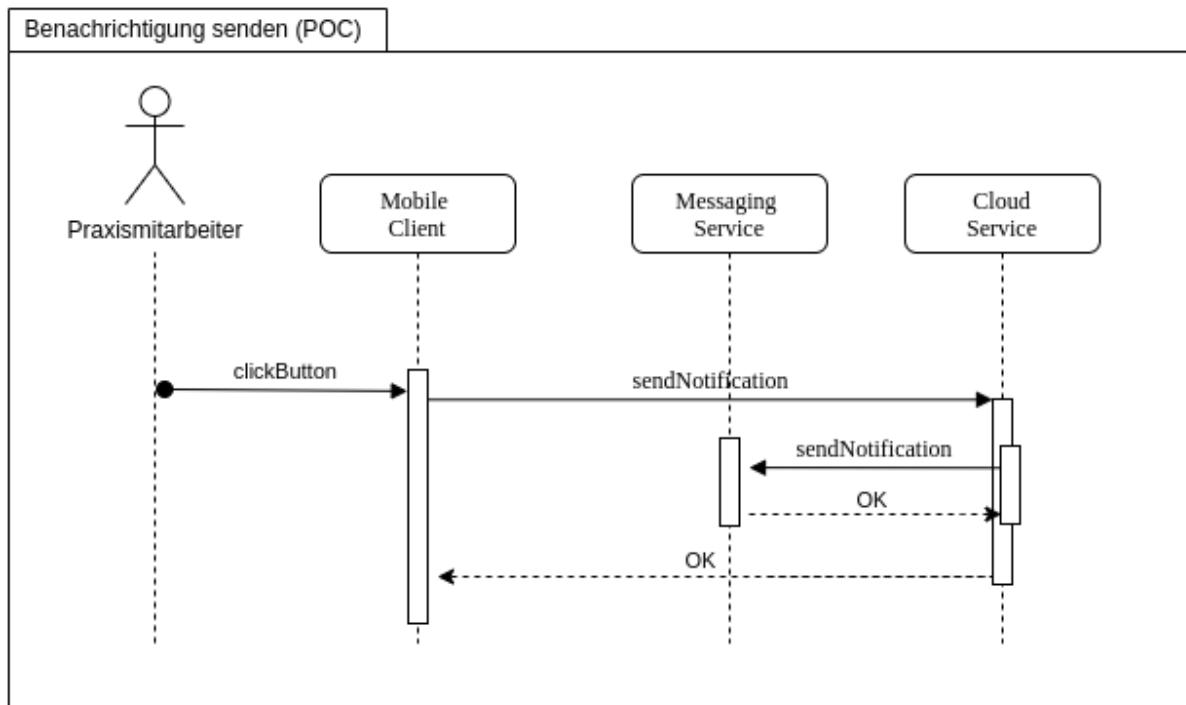


Abbildung 5.20: Proof Of Concept - Benachrichtigung versenden

Wurde eine Benachrichtigung über den Message Service an den Mobile Client versendet, wird diese vom Mobile Client empfangen. Als Reaktion auf den Empfang der Benachrichtigung wird der Inhalt der Benachrichtigung im Mobile Client angezeigt. Die Benachrichtigung löst zudem eine Push-Benachrichtigung auf dem Host Gerät aus.

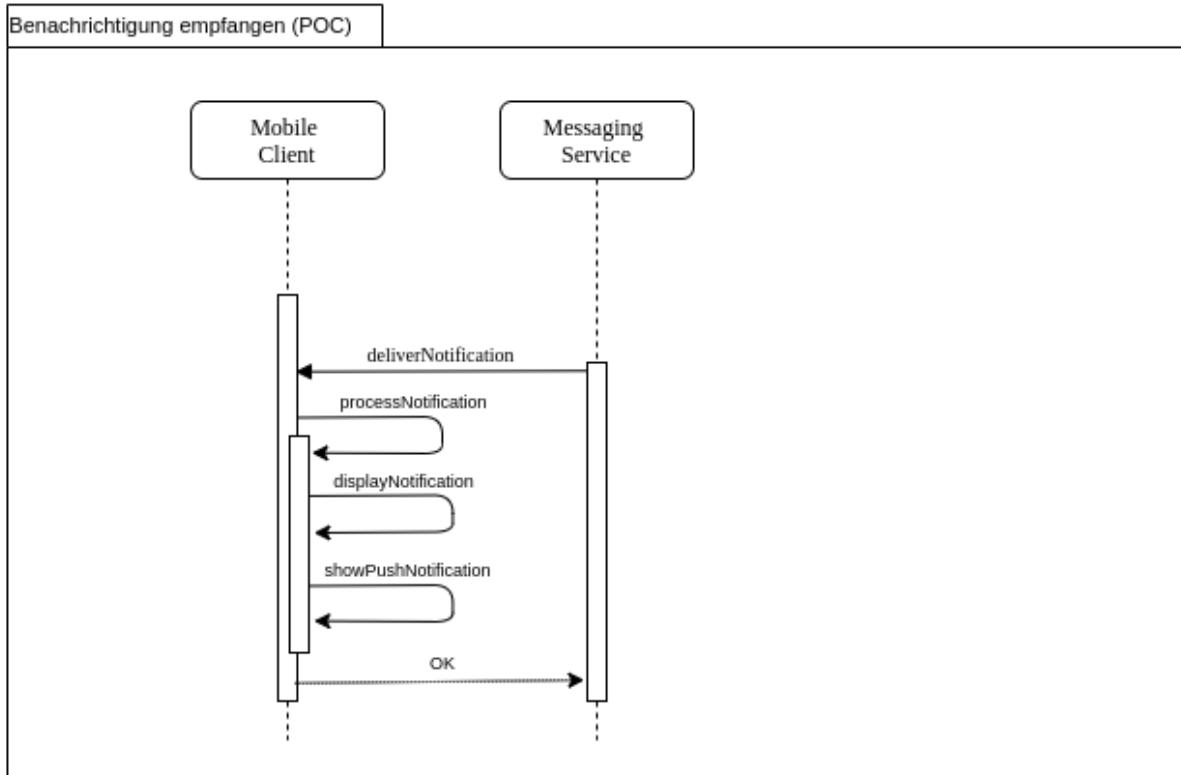


Abbildung 5.21: Proof Of Concept - Benachrichtigung empfangen

6 Umsetzung

6.1 Resultate

Das Praxisrufsystem wurde wie im Kapitel 5 - Konzept beschrieben umgesetzt. Es wurden die drei Komponenten Mobile Client, Cloud Service und Admin UI implementiert. Zudem wurde Firebase Cloud Messaging als Messaging Service angebunden, um Benachrichtigungen zwischen Mobile Clients zu versenden. Der Mobile Client kann dabei verwendet werden, um Benachrichtigungen zu versenden und empfangen. Cloud Service und Admin UI ermöglichen die Konfiguration für Versand und Empfang von Benachrichtigungen zu verwalten und anzuwenden. Weiter wurde mit Amazon Web Services (AWS) eine CI/CD Umgebung aufgebaut, die es erlaubt Cloud Service und das Admin UI zu betreiben und testen. Diese Umgebung wird dem Kunden als Template dienen, wie er das Praxisrufsystem in der Praxis betreiben kann⁷.

Im Rahmen des Projektes wurden damit die Meilensteine M01 bis M06⁸ erreicht. Umgesetzt wurden die Meilensteine mit den folgenden User Stories⁹ inklusive aller dazu definierten Features und Szenarien¹⁰:

- U01 - Benachrichtigung versenden
- U02 - Benachrichtigungen empfangen
- U03 - Nur relevante Benachrichtigungen empfangen
- U04 - Auf Benachrichtigungen aufmerksam machen
- U05 - Verpasste Benachrichtigungen anzeigen
- U06 - Fehler beim Versenden von Benachrichtigungen anzeigen
- U07 - Konfiguration auf Mobile Client auswählen
- U12 - Mehrere Mobile Clients konfigurieren
- U13 - Individuelle Konfiguration pro Mobile Client
- U14 - Zentrale Konfigurationsverwaltung
- T01 - Mobile Client unterstützt iPads
- T02 - Mobile Client unterstützt Android Tablets
- T03 - Geteilte Code Basis für Android und iOS
- T04 - Betrieb mit AWS

Die Umsetzung der CI/CD Pipeline sowie das Erstellen der Konzepte für den Mobile Client, haben mehr Zeit in Anspruch genommen, als geplant war. Aufgrund dieser Verzögerungen konnten die Meilensteine M06 bis M10 nicht umgesetzt werden. Dementsprechend wurden keine Anforderungen dazu umgesetzt. Dies betrifft die User Stories U08 bis U16¹¹:

- U08 - Physicher Knopf am Behandlungsstuhl
- U09 - Text To Speech für Benachrichtigungen
- U10 - Direkte Unterhaltungen zwischen Mobile Clients
- U11 - Gruppenunterhaltungen zwischen Mobile Clients
- U15 - Konfiguration von direkten Anrufen
- U16 - Konfiguration von Gruppenanrufen

⁷Siehe Anhang D

⁸Siehe Kapitel 2.2

⁹Siehe Kapitel 3

¹⁰Siehe Anhang E

¹¹Siehe Anhang E

6.1.1 Mobile Client

Dieses Kapitel zeigt die umgesetzten Ansichten des Mobile Clients. Weitere Informationen zur Bedienung des Mobile Clients sind dem Benutzerhandbuch zu entnehmen¹².

Anmeldung und Konfiguration

Wird die Mobile Client Applikation zu ersten Mal geöffnet, muss die korrekte Konfiguration geladen werden. So können Buttons für die benötigten Benachrichtigungen angezeigt und relevante Benachrichtigungen empfangen werden. In einem ersten Schritt wird dem Benutzer deshalb eine einfache Login-Maske angezeigt. Darin kann sich der Benutzer mit Benutzernamen und Passwort anmelden. War die Anmeldung erfolgreich, werden alle Konfigurationen geladen, die dem Benutzer zur Verfügung stehen. Die verfügbaren Konfigurationen werden dem Benutzer in einer Liste angezeigt und er wird aufgefordert, die gewünschte Konfiguration auszuwählen. Nachdem die Auswahl erfolgt ist, wird der Benutzer zur Startseite weitergeleitet.

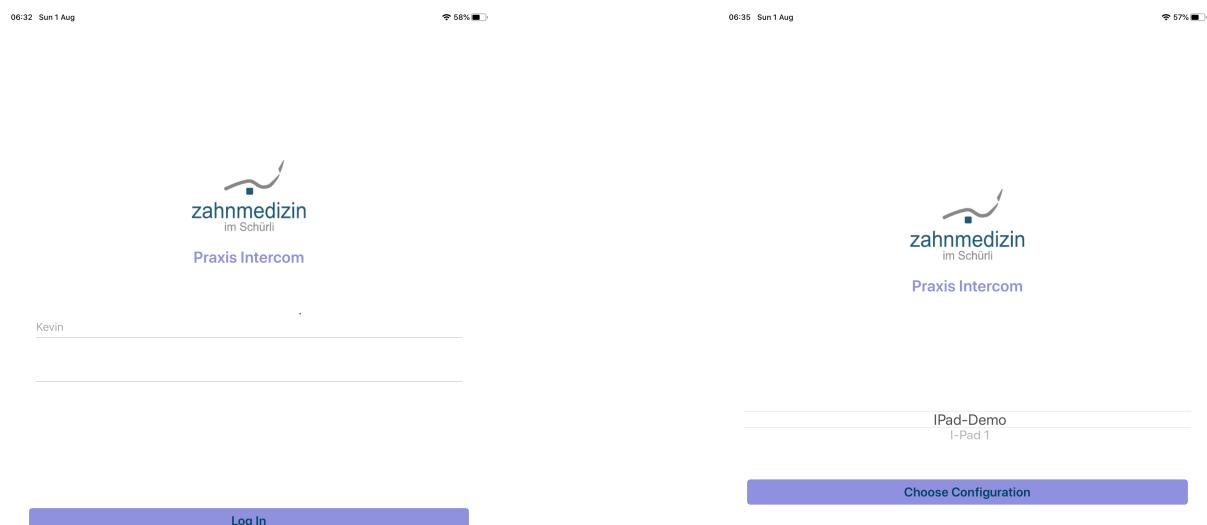


Abbildung 6.1: Login

Abbildung 6.2: Konfiguration

¹²Siehe Anhang C

Benachrichtigungen versenden

Im Tab Home der Startseite werden Buttons angezeigt, um Benachrichtigungen zu versenden. Die Buttons werden dynamisch aus der geladenen Konfiguration generiert. Dabei gibt die Konfiguration den Text vor, der auf dem Button angezeigt wird. Der Inhalt der Benachrichtigung, die der Button auslöst, ist in der Konfiguration im Cloud Service hinterlegt.

Tippt der Benutzer auf einen der Buttons, wird die entsprechende Benachrichtigung versendet. Die Vermittlung, an die relevanten Empfänger übernimmt dabei der Cloud Service. Dieser entscheidet anhand der vorhandenen Konfiguration, welchen Clients die Benachrichtigung zugestellt wird. Schlägt das Versenden der Benachrichtigung an mindestens einen Empfänger fehl, wird dies dem Benutzer angezeigt. Der Benutzer hat dann die Möglichkeit, das Versenden an diese Empfänger wiederholen.

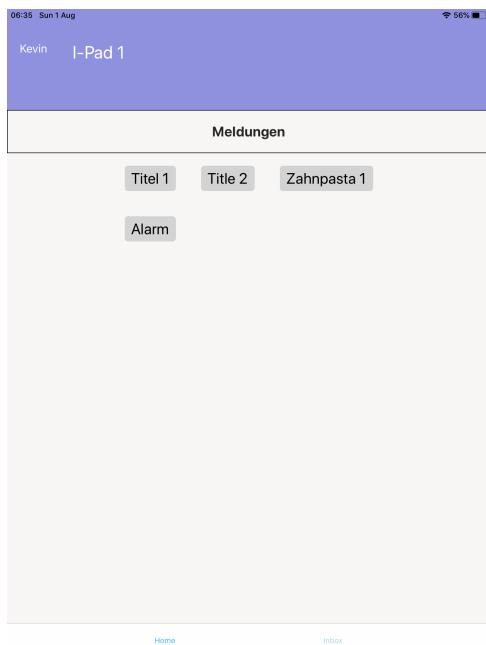


Abbildung 6.3: Home

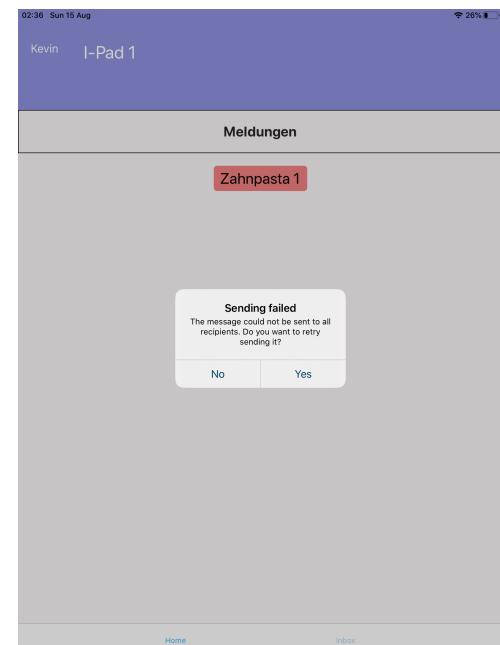


Abbildung 6.4: Retry

Benachrichtigungen empfangen

Wurde eine Benachrichtigung empfangen, ertönt ein Audio Signal und die Benachrichtigung ist im Tab Inbox auf der Startseite ersichtlich. Durch Klick auf einen der Einträge in der Liste, kann der Benutzer die empfangene Benachrichtigung quittieren. Wenn die Inbox Benachrichtigungen enthält, die nicht quittiert wurden, wiederholt der Client im Abstand von 30 Sekunden. Die Quittierung von Benachrichtigungen erfolgt nur lokal auf dem Gerät. Der Versender wird nicht über die Quittierung benachrichtigt. Wurde eine Benachrichtigung im Hintergrund empfangen, wird diese als Push-Benachrichtigung auf dem Gerät angezeigt. Auch wenn die Benachrichtigung im Hintergrund empfangen wurde, wird diese in der Inbox angezeigt.

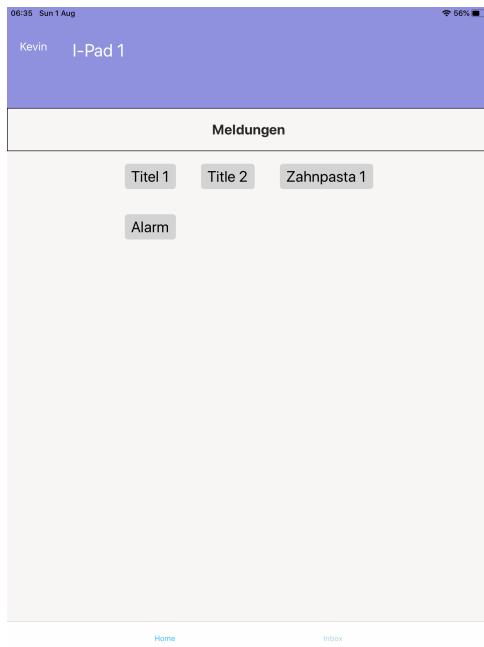


Abbildung 6.5: Inbox

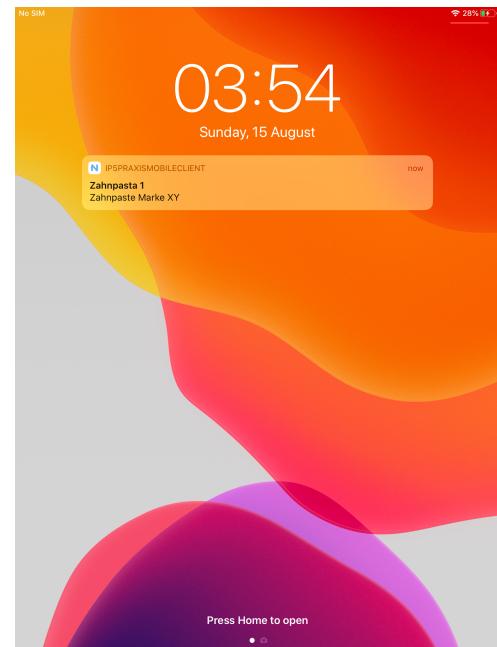


Abbildung 6.6: Push Benachrichtigung

6.1.2 Cloud Service

Der Cloud Service wurde wie im Konzept beschrieben umgesetzt. Er dient dazu, die Konfiguration des Praxisrufsystems persistent zu verwalten und Benachrichtigungen von Mobile Clients entgegenzunehmen. Empfangene Benachrichtigungen werden gemäss der persistierten Konfiguration über den angebundenen Message Service an Mobile Clients weitergeleitet.

Um diese Aufgaben zu erfüllen, bietet der Cloud Service eine REST Schnittstelle an, welche die benötigten Funktionen zugänglich macht. Die umgesetzte API ist unter <https://www.praxisruf.ch/swagger-ui.html> dokumentiert. Die zugehörige Open API definition ist zudem zusammen mit dem Quellcode des Cloud Services abgelegt¹³. Dies Beinhaltet die verfügbaren Endpunkte und das Model welches für Input und Output Werte dieser Schnittstelle dienen.

The screenshot shows the Swagger UI interface for the Cloud Service API. At the top, there is a green header bar with the word "swagger" and a dropdown menu labeled "Select a spec" set to "default". Below the header, the title "Api Documentation" is displayed with a small "1.0" badge next to it. A note indicates the "Base URL: www.praxisruf.ch/" and the "https://www.praxisruf.ch/v2/api-docs" endpoint. Below this, there are links to "Api Documentation", "Terms of service", and "Apache 2.0". The main content area lists various API endpoints under different controllers:

- Client** Clients Controller
 - Client Configuration** Client Configuration Controller
 - Notification** Notification Controller
 - Notification Types** Notification Type Controller
 - Registration** Registrations Controller
 - Users** Users Controller
 - basic-error-controller** Basic Error Controller
- Models**

Abbildung 6.7: Swagger UI

¹³Siehe Anhang B

6.1.3 Admin UI

Das Admin UI bietet Praxisverantwortlichen die Möglichkeit die Konfiguration des Praxisrufsystems zu verwalten. Da nur Praxisverantwortliche das Admin UI verwenden dürfen, ist die Benutzeroberfläche durch ein Login geschützt. Die entsprechenden Anmeldeinformationen müssen bei Installation des Cloud Services vom Betreiber manuell konfiguriert werden.¹⁴

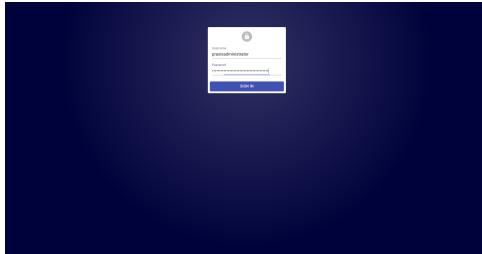


Abbildung 6.8: Login

Configurations					
	Name	Regel Parameter	Notification Type		
<input type="checkbox"/> Client	Emergency	Regel Parameter: Value: (Client)	Notification Type: (None) (Emergency) (Alert) (Info)		
<input type="checkbox"/> Client	Normal	Regel Parameter: Value: (Client)	Notification Type: (None) (Normal)		
<input type="checkbox"/> Client	Feedback	Regel Parameter: Value: (Client)	Notification Type: (None) (Feedback)		

Abbildung 6.9: Configuration Overview

Das Admin UI beinhaltet vier Bereiche. Der Bereich Users dient dazu Benutzer zu erstellen, welche sich als Benutzer am Mobile Client anmelden können. Unter dem Bereich Clients können Geräte verwaltet werden. Jeder Client ist eindeutig einem Benutzer zugewiesen. Im Bereich Notification Types können Benachrichtigungen verwaltet werden. Hier wird konfiguriert, welchen Text der Button für diese Benachrichtigungen im Mobile Client hat und welchen Inhalt die Benachrichtigung hat, wenn sie versendet wird. Unter dem Bereich Configurations wird die zentrale Konfiguration eines Clients verwaltet. Jede Konfiguration wird genau einem Client zugewiesen. Die Konfiguration beinhaltet eine Liste von Notification Types, welche auf dem zugewiesenen Mobile Client als Versenden Button angezeigt werden. Weiter definiert die Konfiguration eine Liste von Regel Parametern, welche bestimmen, welche Benachrichtigungen dem zugewiesenen Client weitergeleitet werden.

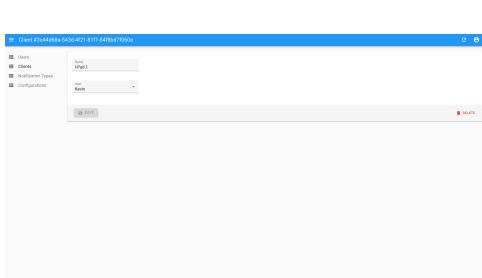


Abbildung 6.10: Login

Notification Types					
	Name	Body	Description		
<input type="checkbox"/> Emergency	Emergency	Body: (Client)	Description: (None) (Emergency) (Alert) (Info)		
<input type="checkbox"/> Normal	Normal	Body: (Client)	Description: (None) (Normal)		
<input type="checkbox"/> Feedback	Feedback	Body: (Client)	Description: (None) (Feedback)		

Abbildung 6.11: Configuration Overview

Jeder der Bereiche im Admin UI bietet dem Benutzer die Möglichkeit, den jeweiligen Teil der Konfiguration zu lesen, erstellen, bearbeiten und löschen. Auf der Startseite jedes Bereiches wird eine Liste mit allen relevanten Einträgen angezeigt. Mehr Informationen zur Bedienung des Admin UIs befinden sich im Benutzerhandbuch.¹⁵

¹⁴Siehe Anhang D

¹⁵Siehe Anhang C

6.2 Tests

Benutzertests

Testablauf

Am 21. Juli 2021 wurden zusammen mit dem Auftraggeber Benutzertests durchgeführt. Dazu wurde der Mobile Client auf ein physisches Ipad installiert. Zusätzlich wurde eine zweite Mobile Client Instanz auf einem Emulator gestartet. Der Cloud Service sowie das Admin UI wurden mit Amazon Webservices deployt. Ein Firebase Messaging Service wurde angebunden.

Für die Benutzertests wurde folgender Ablauf durchgespielt:

1. Client 1 im Admin UI anlegen und Benutzer 1 zuweisen.
2. Client 2 im Admin UI anlegen und Benutzer 1 zuweisen.
3. Einen neuen Notification Type im Admin UI anlegen
4. Eine Client Configuration für Client 1 erstellen und darauf den erfassten Notification Type setzen.
5. Eine Client Configuration für Client 2 erstellen und Regel Parameter erfassen, das alle Benachrichtigungen von Client 1 empfangen werden sollen.
6. Mobile Client auf Ipad starten
7. Auf Ipad mit Benutzer 1 anmelden.
8. Auf Ipad Client 2 auswählen.
9. Mobile Client auf Emulator starten.
10. Auf Emulator mit Benutzer 1 anmelden.
11. Auf Emulator Client 1 auswählen.
12. Auf Emulator Benachrichtigung auslösen.

Der Test wurde zweimal durchgeführt. Bei der ersten Durchführung war die Mobile Client Applikation auf dem Empfänger Gerät im Vordergrund geöffnet. Bei der zweiten Durchführung wurde die Mobile Client Applikation auf dem Empfänger Gerät minimiert. In beiden Fällen wurde erwartet, dass die Benachrichtigung in der Inbox des Empfängers angezeigt wird. Im zweiten Fall wurde zusätzlich erwartet, dass eine Push-Benachrichtigung angezeigt wird.

Mehr oder frühere Benutzertests konnten aufgrund der pandemischen Situation nicht durchgeführt werden.

Feedback vom Benutzer

Aus den Tests mit dem Benutzer sind folgende zusätzliche Anforderungen hervorgegangen:

- Wenn Benachrichtigungen eingehen sollte ein Audiosignal ertönen.
- Wenn Benachrichtigungen nicht quittiert werden, soll ein Erinnerungston ertönen.
- Wenn Benachrichtigungen quittiert werden, soll der Versender darüber informiert werden.

Um diesen Wünschen Gerecht zu werden, wurden die Szenarien S07 und S09 hinzugefügt.¹⁶ Diese Anforderungen konnten im Rahmen des Projektes umgesetzt werden. Der dritte Wunsch des Kunden, die Quittierung von Benachrichtigungen an den Sender weitergeleitet wird konnte aus zeitlichen Gründen nicht mehr umgesetzt werden.

¹⁶Siehe Anhang E

Testplan

Aus den vervollständigten Features und Szenarien wurde der finale Testplan für das umgesetzte System definiert. Alle Szenarien sind detailliert im Anhang E beschrieben. Jedes Szenario definiert die erwarteten Vorbedingungen, einen Testschritt und die zu verifizierenden Ergebnisse. Diese Szenarien dienen als Grundlage für den Testplan, wobei jedes Szenario einen Testfall darstellt.

Folgendes Protokoll zeigt den Stand der letzten Ausführung der Tests am 19.08.2021:

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	+

6.3 Fazit

In diesem Kapitel werden die zentralen Herausforderungen während der Projektarbeit und die Schlussfolgerungen die wir daraus ziehen beschrieben.

Die grösste Herausforderung im Projekt waren Entwicklung und Konzept des Mobile Clients mit einer geteilten Codebasis für iOS und Android. Die Recherchen und Tests in diesem Bereich haben deutlich mehr Zeit in Anspruch genommen, als ursprünglich geplant. Mit der gewählten Technologie Native Script konnten die Anforderungen an den Mobile Client schlussendlich umgesetzt werden. Grosse Teile des Mobile Clients konnten für Android und iOS gleichzeitig umgesetzt werden. An den Stellen wo die Applikation mit dem Betriebssystem interagieren muss, müssen aber Unterschiede gemacht werden und Betriebssystemspezifische Plugins verwendet werden. Die Dokumentation des Frameworks zeigt, das es eine Vielzahl von Plugins gibt und somit fast alle Funktionen die eine native Applikation bietet auch mit Native Script umgesetzt werden können. Während der Entwicklung mussten wir jedoch feststellen, dass diese Anbindungen in der Praxis oft nicht wie erwartet funktionieren. Entweder, weil Plugins unvollständig dokumentiert und implementiert sind oder, weil ein Plugin nur mit einer spezifischen Version von Nativescript oder iOS funktioniert. Dies zeigt, dass eine geteilte Codebasis für mehrere Plattformen Zeit bei der Entwicklung und Wartung sparen kann. Sie bringt aber den Nachteil, dass die Applikation komplizierter wird, weil trotzdem oft zwischen den unterstützten Betriebssystemen unterschieden werden muss. Weiter erschwert es die Verwendung von Betriebssystemfunktionen und Gerätehardware, da diese für alle Betriebssystem abstrahiert werden müssen. Schlussendlich bringt dies eine zusätzliche Schicht in die Architektur die bei Änderungen am Betriebssystem und am Framework Änderungen benötigt. Dies erhöht den Wartungsaufwand deutlich und kann langfristig Probleme für die Kompatibilität mit dem Betriebssystem verursachen.

Wir schliessen daraus, dass eine geteilte Basis vor allem bei Applikationen die keine oder nur wenig Interaktion mit dem Betriebssystem benötigen, einen Vorteil bietet. Sobald aber kritische Teile der Applikation mit betriebssystemnahen Funktionen zusammenhängen, empfehlen wir mehrere native Applikationen zu entwickeln. Der Nachteil, dass Funktionen doppelt implementiert werden müssen wird durch bessere Zukunftssicherheit und einfachere Anbindung an das Betriebssystem aufgehoben.

Eine weitere Herausforderung war das Konzept für den Cloud Service zu erstellen. Dieser musste ein konfigurierbares Subscription System bieten über den die Mobile Clients Benachrichtigungen versenden und empfangen können. Bedingung dafür war dass, individuell Regeln konfiguriert werden können die an unterschiedlichen Bedingungen prüfen, welche Benachrichtigungen für welche Clients relevant sind. Dabei ist es wichtig, dass das Konzept es ermöglicht in Zukunft mit geringem Aufwand weitere Regeln zu implementieren. Neben dem Regelwerk für Benachrichtigungen war auch die Erweiterbarkeit und Skalierbarkeit des Cloud Services als Ganzes eine Herausforderung. Anfänglich sind wir davon ausgegangen, dass sich der Cloud Service als einfache Applikation mit einem Endpunkt für Konfiguration und einem Endpunkt für Benachrichtigungen umsetzen lässt. Die Entwicklungs- und Konzeptphase haben aber gezeigt, dass es auch innerhalb der Domänen eine saubere Aufteilung braucht. Dementsprechend musste die API aufgeteilt werden, um die Weiterentwicklung und Wartbarkeit des Projektes zu gewähren. Das Aufsetzen der Entwicklungs- und Betriebsstruktur mit Amazon Webservices hat ebenfalls deutlich mehr Zeit als geplant in Anspruch genommen. Diese Projektarbeit hat gezeigt, das AWS alle nötigen Mittel bietet, um ein cloudbasiertes Praxisrufsystem zu betreiben. Damit dies effizient umgesetzt werden kann ist es aber essenziell, ein Konzept zu erstellen, welches die benötigten Dienste und Konfigurationen beschreibt. Das Installationshandbuch im Anhang kann als Vorlage für ein solches Konzept dienen¹⁷.

Wir haben gelernt, dass es zentral ist alle Konzepte bei Projektstart zu erstellen. Die Konzepte müssen dabei nicht von Anfang an ausgereift sein. Sie dienen aber als Startpunkt und Orientierungshilfe im

¹⁷Siehe Anhang D

Projekt. Dabei ist es wichtig, dass auch Betriebsinfrastruktur, Skalierbarkeit und Erweiterbarkeit von Anfang an bedacht werden.

Die letzte Herausforderung, die hier erwähnt werden soll, betrifft das Erarbeiten der Anforderungen. Die Anforderungen und Prioritäten laufend mit dem Auftraggeber zu besprechen hat für die Projektarbeit immer klare nächste Ziele gegeben. Da nicht alle Anforderungen bei Projektanfang vollständig definiert wurden, war es teilweise schwer den aktuellen Stand des Projektes einzuordnen.

Für weitere Projekte in diesem Rahmen empfehlen wir, alle relevanten Anforderungen bei Projektstart so detailliert wie möglich zu erarbeiten. Diese Anforderungen können im Projektverlauf regelmässig besprochen, priorisiert und wenn nötig angepasst werden. Dadurch ist es einfacher den Fortschritt des Projekts zu evaluieren und trotzdem möglich schnell auf neue Erkenntnisse zu reagieren.

7 Schluss

Im Rahmen dieser Projektarbeit konnte ein cloudbasiertes Praxisrufsystem umgesetzt werden. Das umgesetzte System besteht aus einer Mobilen Applikation für iOS und Android, einem Cloud Service und einer Web-Applikation. Mit Firebase Messaging wurde ein externer Messaging Service angebunden. Zudem wurde eine Entwicklungs- und Betriebsinfrastruktur mit Amazon Webservices aufgebaut, welche es erlaubt den Cloud Service und die Web-Applikation zu betreiben.

Das umgesetzte Praxisrufsystem ermöglicht es Benachrichtigungen über eine Mobile Applikation zu versenden. Als Endgeräte können dafür iPads oder Android Tablets verwendet und empfangen werden. Der Mobile Client ermöglicht es dabei Benachrichtigungen auch zu empfangen, wenn die Applikation im Hintergrund läuft und sammelt alle Benachrichtigungen in einer Inbox. Welche Benachrichtigungen versendet werden können, kann über eine Web-Applikation pro Gerät konfiguriert werden. Weiter können über die Web-Applikation Regeln definiert werden, welche Benachrichtigungen ein Gerät empfangen soll. So ist es mit dem Praxisrufsystem möglich vorkonfigurierte Benachrichtigungen an einzelne, mehrere oder alle angebundenen Clients zu versenden.

Das umgesetzte System hat aber durchaus noch Lücken, welche eine produktive Nutzung verhindern könnten. Das Praxisrufsystem konnte wegen diverser Herausforderungen¹⁸ nicht im Umfang wie es in der Aufgabenstellung beschrieben wurde umgesetzt werden. Die Anforderungen Benachrichtigungen, mit einer Text-To-Speech-Funktion auszugeben und eine Gegensprechanlage für Direkt- und Gruppen gespräche wurden weder als Konzept noch in der Praxis umgesetzt. Dementsprechend bietet das umgesetzte Praxisrufsystem nicht in allen Fällen den Funktionsumfang, der für eine produktive Nutzung nötig wäre.

Werden die Funktionen Text-To-Speech und Gegensprechanlage nicht benötigt, könnte das implementierte System grundsätzlich produktiv eingesetzt werden. Mit dem aktuellen Stand des Systems ist allerdings nur die Konfiguration von einfachen Regeln möglich. Wenn in einem produktiven System kompliziertere oder zusammengesetzte Regeln zur Vermittlung von Benachrichtigung benötigt werden, ist das Praxisrufsystem noch nicht einsetzbar. Das System wurde allerdings so konzipiert, dass zusätzliche und kompliziertere Regeln einfach umgesetzt werden können. Eine Erweiterung des Systems in diesem Punkt wäre deshalb mit verhältnismässig kleinem Aufwand möglich.

Die grösste Gefahr für die produktive Nutzung des Systems ist allerdings die umgesetzte mobile Applikation. Die Technologie mit der die Applikation umgesetzt wurde ermöglicht es, eine Codebasis für Android und iOS Geräte zu teilen. Dies ermöglicht in einigen Bereichen schnellere Entwicklung und einfache Wartung. Es erschwert allerdings die Anbindung von Betriebssystemfunktionen und Zugriff auf Gerätehardware wie es für eine Gegensprechanlage und Text-To-Speech Funktionen nötig wäre.

Insgesamt sind wir zufrieden mit den Konzepten und Ergebnissen, die aus dieser Arbeit hervorgegangen sind. Wir sind überzeugt mit dem Konzept für die Registrierung von Mobile Clients und dem Regelwerk für Benachrichtigungen eine erweiterbare, skalierbare Grundlage geschaffen zu haben. Das umgesetzte Rufsystem bietet eine solide Grundlage um ein konkurrenzfähiges, modernes Praxisrufsystem zu entwickeln, welches alle nötigen Anforderungen für den produktiven Einsatz abdeckt.

¹⁸Siehe Kapitel 6

Literaturverzeichnis

- [1] D. Jossen, *21FS-IMVS01: Cloudbasiertes Praxisrufsystem*, 2020.
- [2] JetBrains. (15. Aug. 2021). Connect to platform-specific APIs, Adresse: <https://kotlinlang.org/docs/mobile/connect-to-platform-specific-apis.html>.
- [3] P. L. Sam Richard. (15. Aug. 2021). What are Progressive Web Apps? Adresse: <https://web.dev/what-are-pwas/>.
- [4] A. Deveria. (15. Aug. 2021). Push Api, Adresse: <https://caniuse.com/push-api>.
- [5] C. Love. (15. Aug. 2021). Progressive Web Applications (PWA) on iOS 13–14 Provide a Rich Channel to Reach Customers Despite the Platform Limitations, Adresse: <https://love2dev.com/pwa/ios/>.
- [6] Apache Software Foundation. (15. Aug. 2021). Overview, Adresse: <https://cordova.apache.org/docs/en/10.x/guide/overview/index.html>.
- [7] Stackshare. (15. Aug. 2021). Apache Cordova vs NativeScript, Adresse: <https://stackshare.io/stackups/apache-cordova-vs-nativescript>.
- [8] OpenJS Foundation. (24. Juli 2021). How NativeScript Works, Adresse: <https://v7.docs.nativescript.org/core-concepts/technical-overview>.
- [9] Flutter. (15. Aug. 2021). Flutter architectural overview, Adresse: <https://flutter.dev/docs/resources/architectural-overview>.
- [10] M. Long. (15. Aug. 2021). Why Flutter Isn't the Next Big Thing, Adresse: <https://betterprogramming.pub/why-flutter-isnt-the-next-big-thing-e268488521f4>.
- [11] OpenJS Foundation. (16. Aug. 2021). Setting up your system, Adresse: <https://docs.nativescript.org/environment-setup.html>.
- [12] Apple Inc. (20. Aug. 2021). Registering Your App with APNs, Adresse: https://developer.apple.com/documentation/usernotifications/registering_your_app_with_apns.
- [13] farfromrefuge. (12. Aug. 2021). NativeScript Push-Benachrichtigungen, Adresse: <https://market.nativescript.org/plugins/nativescript-push/>.
- [14] Mozilla Developer Network. (12. Aug. 2021). EventSource, Adresse: <https://developer.mozilla.org/en-US/docs/Web/API/EventSource>.
- [15] IBM Education. (20. Aug. 2021). Message Brokers, Adresse: <https://www.ibm.com/cloud/learn/message-brokers#:~:text=A%20message%20broker%20is%20software,messages%20between%20formal%20messaging%20protocols.&text=This%20facilitates%20decoupling%20of%20processes%20and%20services%20within%20systems..>
- [16] Google Developers. (12. Aug. 2021). Your server environment and FCM, Adresse: <https://firebase.google.com/docs/cloud-messaging/server>.
- [17] VMWare Inc. (12. Aug. 2021). Why Spring?, Adresse: <https://spring.io/why-spring>.
- [18] J. Villa. (12. Aug. 2021). Spring Boot with AWS, Adresse: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html>.
- [19] OpenJS Foundation. (24. Juli 2021). What is iOS Runtime for NativeScript?, Adresse: <https://v7.docs.nativescript.org/core-concepts/ios-runtime/overview>.
- [20] S. Odean. (1. Aug. 2021). Software Architecture — The Onion Architecture, Adresse: <https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec>.
- [21] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns*. Addison-Wesley, 1994.
- [22] Spring Io. (16. Aug. 2021). BasicAuthentication, Adresse: <https://docs.spring.io/spring-security/site/docs/current/reference/html5/#servlet-authentication-basic>.

- [23] S. Hsu. (16. Aug. 2021). Session vs Token Based Authentication, Adresse: <https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4>.
- [24] Auth0 Inc. (16. Aug. 2021). What is JSON Web Token, Adresse: <https://jwt.io/introduction>.
- [25] Marmelab. (1. Aug. 2021). react-admin, Adresse: <https://marmelab.com/react-admin/Readme.html>.
- [26] Google Developers. (14. Aug. 2021). Understand Firebase Projects, Adresse: https://firebase.google.com/docs/projects/learn-more#setting_up_a_firebase_project_and_registering_apps.
- [27] OpenJS Foundation. (16. Aug. 2021). CLI Basics, Adresse: <https://docs.nativescript.org/development-workflow.html>.
- [28] Amazon Web Services. (14. Aug. 2021). AWS Amplify - Getting started with existing code, Adresse: <https://docs.aws.amazon.com/amplify/latest/userguide/getting-started.html>.
- [29] ——, (14. Aug. 2021). AWS Amplify - Set up custom domains, Adresse: <https://docs.aws.amazon.com/amplify/latest/userguide/custom-domains.html>.
- [30] ——, (14. Aug. 2021). AWS Elastic Beanstalk - Getting Started, Adresse: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.CreateApp.html>.
- [31] ——, (14. Aug. 2021). AWS Elastic Beanstalk - Adding a database to your Elastic Beanstalk environment, Adresse: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html>.
- [32] ——, (14. Aug. 2021). AWS Elastic Beanstalk - Environment properties and other software settings, Adresse: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environments-cfg-softwaresettings.html>.
- [33] ——, (14. Aug. 2021). How can I configure HTTPS for my Elastic Beanstalk environment? Adresse: <https://aws.amazon.com/premiumsupport/knowledge-center/elastic-beanstalk-https-configuration/>.
- [34] ——, (14. Aug. 2021). Connecting to a DB instance running the PostgreSQL database engine, Adresse: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToPostgreSQLInstance.html.

Abbildungsverzeichnis

2.1	Projektplan	2
5.1	Systemkomponenten	9
5.2	NativeScript-Overview	10
5.3	Mobile-Client Package Diagramm	13
5.4	Mobile-Client Flow Chart	14
5.5	HomeScreen Mockup	15
5.6	Inbox Mockup	15
5.7	Onion Architecture	16
5.8	Package Struktur Cloud Service	17
5.9	Domänenmodell Configuration	18
5.10	Klassendiagramm Configuration Service Interfaces	19
5.11	Klassendiagramm Rules Engine	20
5.12	Domänenmodell Notification	21
5.13	Klassendiagramm Notification Service Interfaces	21
5.14	Ablauf Registration	22
5.15	Ablauf Benachrichtigung Senden und Empfangen	23
5.16	Ablauf Benachrichtigung Wiederholen	24
5.17	Authentifizierung-Sequenz	26
5.18	SecurityContext-Klassendiagramm	27
5.19	Admin-Ui Package Diagramm	28
5.20	Proof Of Concept - Benachrichtigung versenden	30
5.21	Proof Of Concept - Benachrichtigung empfangen	31
6.1	Login	33
6.2	Konfiguration	33
6.3	Home	34
6.4	Retry	34
6.5	Inbox	35
6.6	Push Benachrichtigung	35
6.7	Swagger UI	36
6.8	Login	37
6.9	Configuration Overview	37
6.10	Login	37

6.11 Configuration Overview	37
A.1 Aufgabenstellung	46

A Aufgabenstellung

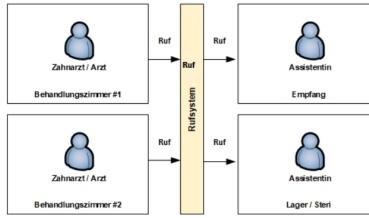


21FS_IMVS01: Cloudbasiertes Praxisrufsystem

Betreuer:	<u>Daniel Jossen</u>	Priorität 1	Priorität 2
Arbeitsumfang:	P5 (180h pro Student)	---	---
Teamgrösse:	2er Team	---	---
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 Gegengesprechfunktion zu integrieren. Ein durchgeführte Marktanalyse hat gezeigt, dass die meisten auf dem Markt kommerziell erhältlichen Rufsysteme auf proprietären Standards beruhen und ein veraltetes Busystem 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:

- Definition und Entwicklung einer skalierbaren Softwarearchitektur
- App ist auf Android und IOS basierten Tablets einsetzbar
- App besitzt eine integrierte Gegesprechfunktion (1:1 oder 1:m Kommunikation)
- Auf der App können beliebige Buttons konfiguriert werden, die anschliessend auf den anderen Tablets einen Alarm oder eine Meldung (Text- oder Sprachmeldung) generieren
- Textnachrichten können als Sprachnachricht (Text to Speech) ausgegeben werden
- Verschlüsselte Übertragung aller Meldungen zwischen den einzelnen Stationen
- Integration der Lösung in den Zahnarztbehandlungsstuhl über einen Raspberry PI
- Offene API für Integration in Praxisadministrationssystem

Problemstellung

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

Technologien/Fachliche Schwerpunkte/Referenzen

- Cloud Services
- Skalierbare Softwarearchitektur
- Text to Speech Funktion
- Android und IOS App-Entwicklung
- Sichere Übertragung von Sprach- und Textmeldungen

Abbildung A.1: Aufgabenstellung

B Quellcode

Sämtlicher Quellcode der im Rahmen des Projektes entsteht, wurde mit Git verwaltet. Der Quellcode ist für Berechtigte unter dem Projekt IP5-Cloudbasiertes-Praxisrufsystem auf [github.com](https://github.com/IP5-Cloudbasiertes-Praxisrufsystem) einsehbar¹⁹. Berechtigungen können bei Joshua Villing oder Kevin Zellweger angefordert werden.

¹⁹<https://github.com/IP5-Cloudbasiertes-Praxisrufsystem>

C Benutzerhandbuch

Mobile Client

Anmeldung und Konfiguration

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

Benachrichtigung versenden

1. In Mobile Client anmelden
2. In Navigationsleiste (unten) "Home" auswählen
3. Button mit dem gewünschten Titel antippen
4. Die Benachrichtigung wird versendet

Benachrichtigung Auflisten und Quittieren

1. In Mobile Client anmelden
2. In Navigationsleiste (unten) "Inbox" auswählen
3. In dieser Ansicht sehen Sie alle empfangenen noch nicht quittierten Benachrichtigungen
4. Zum Quittieren einer Benachrichtigung den "OK" Button der antippen

Abmeldung

1. Mobile Client Applikation öffnen
2. Auf den Namen in der Kopfleiste tippen
3. Logout bestätigen

Admin UI

Anmeldung

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

Einträge hinzufügen

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

Praxisruf konfigurieren

1. Melden Sie sich im Admin UI an.
2. Erfassen Sie in der Kategorie "User" mindestens einen Benutzer.
3. Erfassen Sie in der Kategorie "Client" pro Endgerät in Ihrem System einen Eintrag und weisen Sie ihn dem Benutzer des entsprechenden Zimmers zu.

4. Erfassen Sie in der Kategorie “Notification Types” alle Benachrichtigungen die Sie zur Verfügung stellen möchten.
5. Erfassen Sie in der Kategorie “Configurations” pro Gerät und Zimmer einen Eintrag und weisen es dem entsprechenden Client zu.
6. Unter Notification Types können die Benachrichtigungen auswählen, die auf diesem Gerät zum Versenden verfügbar sind.
7. Unter Rule Parameters können Sie Regeln definieren, welche Benachrichtigungen diesem Client zugestellt werden.
 - Mit dem Rule Type “Sender”, werden alle Benachrichtigungen vom angegebenen Sender Empfangen.
 - Mit dem Rule Type “NotificationType”, werden alle Benachrichtigungen mit dem angegebenen Typ empfangen.

D Installationsanleitung

Firebase Messaging

1. Öffnen Sie die *Firebase Console* und erstellen Sie ein neues Projekt.
2. Navigieren Sie in das Menü “Project Settings”.
3. Navigieren Sie zum Tab “Service Accounts”.
4. Klicken Sie den Button “Generate new private Key” und bestätigen Sie die Eingabe.
5. Speichern Sie die generierte Datei an einem Ort Ihrer Wahl.
6. Kopieren Sie den Dateiinhalt und Benutzen Sie ein Werkzeug Ihrer Wahl²⁰ um den Inhalt mit Base64 zu kodieren.
7. Speichern Sie den Base64 kodierten Dateiinhalt an einem Ort Ihrer Wahl. Sie werden diesen Wert für die Konfiguration des Cloud Services benötigen.
8. Navigieren Sie in das Menü “Project Settings”.
9. Erstellen Sie für die gewünschte Plattform (iOS, Android oder Web app) eine Anwendung via “Add App”.
10. Verwenden Sie für den Android package name bzw. die iOS bundle ID den package_name des Mobile Clients. (ch.fhnw.ip5.praxisintercom.client)
11. Vergeben Sie einen beliebigen App-Nickname.
12. Laden Sie nach dem Registrieren die Konfigurationsdatei GoogleService-Info.plist (iOS) oder google-services.json (Android) herunter.
13. Legen Sie die Konfigurationsdatei im MobileClient Projekt unter ./App_Resources/<Plattform>ab.

Mehr Informationen zu Firebase und der Integration von Firebase Projekten finden Sie in der offiziellen Dokumentation.[26]

Mobile Client

Installation des Bestehenden Images

Um den Aktuellen Stand des Mobile-Clients direkt auf einem IPad zu installieren folgen Sie folgenden Schritten:

1. Stellen Sie sicher, dass Sie Zugang zu einem MacOS Gerät mit Installiertem Xcode haben.
2. Verbinden Sie das gewünschte IPad per USB mit Ihrem Gerät.
3. Öffnen Sie die Deviceliste innerhalb von Xcode. (cmd + shift + 2)
4. Installieren Sie das unter <PathToMobileClientRepo>/ReleaseBuild abgelegte .ipa File via Drag and Drop auf dem IPad.

Installation Entwicklungsumgebung

1. Stellen Sie sicher, dass Sie Node und NPM auf Ihrem System installiert haben.
2. Folgen Sie dem *Setup Guide* von NativeScript für Ihre Systemkonfiguration [11]
3. Folgen Sie Ihrem Bedarf entsprechend dem *CLI-Workflow*[27]

Release Build Bauen

Um vom mobile Client ein .ipa File zu generieren, welches Sie direkt auf auf ein IPad installieren können, führen Sie im Mobile-Client Projekt folgenden befehl aus: *ns build ios –release –forDevice*

Das gebaute Projekt wird unter <PathToMobileClientRepo>/platforms/ios/build/Release-iphoneos/IP5praxismobileclient erstellt.

²⁰z.B. <https://www.base64decode.org/>

1. Stellen Sie sicher, dass Sie Node und NPM auf Ihrem System installiert haben.
2. Folgen Sie dem *Setup Guide* von NativeScript für Ihre Systemkonfiguration [11]
3. Folgen Sie ihrem Bedarf entsprechend dem *CLI-Workflow*[27]

Admin UI

Im Folgenden wird beschrieben wie die Admin UI Applikation mit AWS betrieben werden kann.

1. AWS Amplify Service aufsetzen:
 - (a) Amazon Webservice unterstützt die Anbindung von Github, Gitlab, BitBucket und AWS CodeCommit. Stellen Sie sicher, dass der Quellcode der Admin UI Applikation in einem Git Repository zur Verfügung steht.
 - (b) Folgen Sie den Schritten in der *offiziellen Anleitung*.[28]
2. Verbindung zum Cloud Service konfigurieren:
 - (a) Öffnen sie die AWS Amplify Konsole für die in Schritt 1 erstellte Applikation.
 - (b) Wählen Sie den Menüpunkt “Environment Variables”
 - (c) Erstellen Sie eine neue Variable mit dem Namen REACT_APP_BACKEND_BASE_URI. Setzen Sie als Wert dafür die Domain, unter welcher der Cloud Service erreichbar ist.
3. Konfigurieren Sie eine Domain für die Admin UI Applikation.
 - (a) Folgen Sie dazu den Schritten in der *offiziellen Anleitung*.[29] folgen.

Cloud Service

Im Folgenden wird beschrieben wie die Cloud Service Applikation mit AWS betrieben werden kann.

1. Stellen Sie sicher, dass der Quellcode der Cloud Service Applikation in einem Git Repository bei einem der Anbieter Github, Gitlab, BitBucket oder AWS CodeCommit zur Verfügung steht.
2. Erstellen Sie einen mit AWS ein Elastic Beanstalk Environment.
 - (a) Folgen Sie dazu der *offiziellen Anleitung*[30]
 - (b) Wählen Sie unter Plattform “Java” und die dazugehörigen Standardeinstellungen.
 - (c) Wählen Sie unter Application Code “Sample Application”.
3. Erstellen Sie mit AWS RDS eine Datenbank für die Cloud Service Applikation
 - (a) Öffnen Sie die Beanstalk Console.
 - (b) Folgen Sie der *offiziellen Anleitung*[31] um eine Relationale Datenbank an das Beanstalk Environment anzubinden.
 - (c) Wählen Sie als Datenbank Engine “postgres” in der Version 13.3.
 - (d) Wenn sie oben genannter Anleitung folgen, ist keine weitere Konfiguration für die Datenbankanbindung im Cloud Service nötig. Sollten Sie wählen, die Datenbank auf eine andere Art zu Betreiben müssen in Schritt 4 die Umgebungsvariablen RDS_HOSTNAME, RDS_PORT, RDS_DB_NAME, RDS_USERNAME und RDS_PASSWORD mit den entsprechenden Werten konfiguriert werden.
4. Definieren Sie die nötigen Umgebungsvariablen für die Cloud Service Applikation:
 - (a) Folgen Sie der *offiziellen Anleitung*[32] um die nötigen Umgebungsvariablen zu setzen:
 - (b) Name: FCM_CREDENTIALS, Wert: Firebase Credentials mit Base 64 Encoded²¹
 - (c) Name: SPRING_PROFILES_ACTIVE, Wert: aws.
 - (d) Name: ADMIN_ORIGIN, Wert: Admin UI Domain
 - (e) Name: JWT_SECRET_KEY, Wert: Zufälliger 64 Bit String²²
5. Konfigurieren Sie AWS CodeBuild um die Cloud Service Applikation mit AWS bauen zu können.
 - (a) Öffnen Sie die AWS Console und wählen Sie unter Services “Code Pipeline” aus.

²¹Siehe Installationsanleitung Firebase Messaging

²²<https://www.grc.com/passwords.htm>

- (b) Wählen Sie die Option “Create build project”.
 - (c) Geben Sie unter Project Configuration einen passenden Namen ein.
 - (d) Wählen Sie unter Source den gewünschten Anbieter und geben Sie die Repository URL sowie die gewünschte Version (Branch Name) ein.
 - (e) Wählen Sie unter Buildspec “Use a buildspec file”. Die projektspezifische Build Konfiguration ist in der Cloud Service Applikation bereits enthalten.
 - (f) Bestätigen Sie die Eingaben.
6. Konfigurieren Sie AWS CodePipeline um die Cloud Service Applikation zu installieren:
- (a) Öffnen Sie die AWS Console und wählen Sie unter Services “Code Pipeline” aus.
 - (b) Wählen Sie die Option “Create New Pipeline”.
 - (c) Geben Sie in Schritt 1 einen Namen für die Pipeline an und wählen Sie “Next”.
 - (d) Wählen Sie in Schritt 2 gewünschten Anbieter und folgen sie dem Wizard um das Git Repository des Cloud Services anzubinden.
 - (e) Wählen Sie in Schritt 3 AWS CodeBuild und das CodeBuild Projekt welches Sie in Schritt 3 erstellt haben.
 - (f) Wählen Sie in Schritt 4 AWS Elastic Beanstalk und die in Schritt 2 definierten Instanzen.
 - (g) Bestätigen Sie die Eingaben.
7. Stellen Sie sicher, dass die installierte Cloud Service Applikation über HTTPS erreichbar ist. Dies ist für die Kommunikation mit Mobile Client Instanzen zwingend notwendig.
- (a) Folgen Sie dazu der offiziellen Anleitung *offiziellen Anleitung*[33].
 - (b) Am Cloud Service sind dabei keine Änderungen nötig.
8. Erstellen Sie einen Administrator Account für das Admin UI.
- (a) Richten Sie den Datenbankzugriff auf die in Schritt 3 erstellte Datenbank gemäss der *offiziellen Anleitung*[34]. ein.
 - (b) Wählen Sie ein Passwort
 - (c) Erstellen Sie ein Hash des gewählten Passwort mit dem ”bcryptAlgorithmus.²³
 - (d) Passen Sie die Parameter in folgenden Script an und führen es auf der Datenbank aus:

```

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

```

Listing 5: createadmin.sql

²³<https://bcrypt-generator.com/>

E Features und Testszenarien

F01 - Benachrichtigungen Versenden

Scenario S01: Benachrichtigung versenden

Given: Benutzer ist vollständig angemeldet
 And: Mindestens ein Empfänger ist konfiguriert
 When: Praxismitarbeiter tippt auf einen Benachrichtigungs-Button
 Then: Benachrichtigung wird an den zentralen Cloud Service gesendet
 And: Benachrichtigung wird an alle Mobile Clients versendet
 die sich für diese Benachrichtigung subscribed haben weitergeleitet
 And: Praxismitarbeiter erhält optische Rückmeldung, dass Benachrichtigung versendet wurde

Scenario S02: Keine Empfänger konfiguriert

Given: Benutzer ist vollständig angemeldet
 And: Kein Empfänger ist konfiguriert
 When: Praxismitarbeiter tippt auf einen Benachrichtigungs-Button
 Then: Benachrichtigung wird an den zentralen Cloud Service gesendet
 And: Benachrichtigung wird nicht weitergeleitet

F02 - Benachrichtigungen Empfangen

Scenario S03: Empfangen

Given: Eine Benachrichtigung wurde von Mobile Client versendet
 When: Cloud Service Notification an Empfänger Mobile Client weiterleitet
 Then: Wird die Benachrichtigung vom Empfänger Mobile Client empfangen
 And: In einer Übersicht für empfangene Benachrichtigung angezeigt.

F03 - Fehlgeschlagene Benachrichtigungen

Scenario S04: Fehler Rückmeldung

Given: Eine Benachrichtigung wurde von Mobile Client versendet
 When: Weiterleitung von Cloud Service Notification an Empfänger schlägt auf Service Seite fehl
 Then: Der Praxismitarbeiter wird über den Fehler informiert
 And: Der Praxismitarbeiter hat die Möglichkeit die fehlgeschlagenen Benachrichtigungen zu wiederholen

Scenario S05: Confirm Retry

Given: Benachrichtigung ist fehlgeschlagen
 And: Dialog zum Wiederholen wird angezeigt
 When: Praxismitarbeiter bestätigt, dass wiederholt werden soll
 Then: Der Cloudservice versucht erneut, die fehlgeschlagenen zuzustellen

Scenario S06: Cancel Retry

Given: Benachrichtigung ist fehlgeschlagen
 And: Dialog zum Wiederholen wird angezeigt
 When: Praxismitarbeiter klickt, dass nicht wiederholt werden soll
 Then: Werden die fehlgeschlagenen nicht wiederholt
 And: Zurück zur Notificationsansicht

F04 - Über Benachrichtigungen Notifizieren

Scenario S07: Foreground

Given: Mobile Client ist geöffnet
 When: Eine Benachrichtigung wird vom Mobile Client empfangen
 Then: Ein Audio Signal erklingt

Scenario S08: Background

Given: Mobile Client läuft im Hintergrund
 When: Eine Benachrichtigung wird vom Mobile Client empfangen
 Then: Ein Audio Signal erklingt
 And: Eine Push Benachrichtigung wird angezeigt

Scenario S09: Nicht Quittiert

Given: Mobile Client ist geöffnet
 And: Eine Benachrichtigung wurde empfangen
 When: Benachrichtigung wird nicht quittiert
 Then: Ein Audio Signal erklingt
 And: Das Audio Signal wiederholt sich alle 30 Sekunden, bis die Benachrichtigung Quittiert wurde.

F05 - Login Mobile Client

Scenario S10: Startbildschirm, wenn nicht angemeldet

Given: Mobile Client ist geöffnet
 When: Benutzer ist nicht angemeldet
 Then: Benutzer wird zum Login aufgefordert

Scenario S11: Startbildschirm, wenn angemeldet

Given: Mobile Client ist geöffnet
 When: Benutzer ist angemeldet
 Then: Konfiguration die der Benutzer zuletzt gewählt hat wird angezeigt
 And: Benachrichtigungs Buttons gemäss Konfiguration werden angezeigt.

Scenario S12: Anmelden korrekt

Given: Benutzer ist nicht angemeldet
 And: Login Screen wird angezeigt
 And: Für den Benutzer sind gültige Konfigurationen erfasst
 When: Benutzer meldet sich mit korrekten Daten an
 Then: Benutzer wird auf nächste Seite geleitet und kann dort die Konfiguration auswählen, die er Benutzen

Scenario S13: Anmelden falsch

Given: Benutzer ist nicht angemeldet
 And: Login Screen wird angezeigt
 When: Benutzer meldet sich mit falschen Daten an
 Then: Fehlermeldung
 And: Benutzer wird nicht weitergeleitet

Scenario S14: Konfiguration Wählen

Given: Benutzer hat sich korrekt angemeldet
 And: Konfiguration Auswählen Screen wird angezeigt
 When: Der Benutzer wählt die gewünschte Konfiguration
 Then: Der Benutzer wird weitergeleitet
 And: Die gewählte Konfiguration wird geladen
 And: Benachrichtigungs Buttons gemäss Konfiguration werden angezeigt.

Scenario S15: Logout

Given: Benutzer ist angemeldet
 When: Benutzer klickt logout
 Then: Benutzer wird zur Login Seite weitergeleitet

F06 - Konfigurationsverwaltung

Scenario S16: Login

Given: Benutzer ist nicht angemeldet
And: Admin UI Login Screen wird angezeigt
When: Admin meldet sich mit korrekten Daten an
Then: Admin wird auf Übersichtsseite weitergeleitet

Scenario S17: Anmelden falsch

Given: Benutzer ist nicht angemeldet
And: Admin UI Login Screen wird angezeigt
When: Admin meldet sich mit falschen Daten an
Then: Fehlermeldung wird angezeigt
And: Admin wird nicht weitergeleitet.

Scenario S18: Konfiguration verwalten

Given: Admin ist angemeldet
When: Admin UI wird aufgerufen
Then: Alle existierenden Konfigurationen werden angezeigt
And: Neue Konfigurationen können erstellt werden
And: Bestehende Konfigurationen können verändert werden
And: Bestehende Konfigurationen können gelöscht werden

F07 - Integration Behandlungsstuhl

Dieses Feature fällt ausserhalb des Projekt Scopes. Dementsprechend wurden dafür noch keine Szenarien definiert.

F08 - Text To Speech

Dieses Feature fällt ausserhalb des Projekt Scopes. Dementsprechend wurden dafür noch keine Szenarien definiert.

F09 - Direkte Anrufe

Dieses Feature fällt ausserhalb des Projekt Scopes. Dementsprechend wurden dafür noch keine Szenarien definiert.

F10 - Gruppen Anrufe

Dieses Feature fällt ausserhalb des Projekt Scopes. Dementsprechend wurden dafür noch keine Szenarien definiert.

F Ehrlichkeitserklärung

«Hiermit erkläre ich, die vorliegende Projektarbeit IP5 - 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 19.08.2021

Unterschrift

Name Kevin Zellweger
Ort Aarau
Datum 19.08.2021

Unterschrift