Fachhochschule Nordwestschweiz
Hochschule für Technik

Cloudbasiertes Praxisrufsystem

IP 5 - Technischer Bericht

14. 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 Praxisruferarbeitet und umgesetzt. Praxisruf ist ein Rufsystem, welches für das Versenden von Benachrichtigungen in einer Praxis verwendet werden kann. 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 einsetzten

Praxisruf könnte bereits heute als produktives Benachrichtigungssystem eingesetzt werden. Im Gegensatz zu vielen bestehenden kann Praxisruf allerdings noch nicht als Gegensprechanlage verwendet werden. Weiter erlaubt die Konfiguration von Nachrichtenempfang 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 verhältnissmässig 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 Praxsiruf 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, sollte dieses 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 zwei 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. Die mobile App 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 grösseren Funktionsumfang als bestehende Lösungen. Dabei ist es aber leicht konfigurierbar und kann somit für die konkreten Bedürfnisse jedes Kunden individualisiert werden.

Inhaltsverzeichnis

1 Einleitung

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

Im Rahmen dieser Arbeit soll ein Cloudbasiertes Praxisrufsystem entwickelt werden. Pro Behandlungszimmer wird ein Android oder IOS basiertes Tablet installiert. Auf diese Tablets 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 Kommunikaion)
- 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 Mutlicast 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 Proejktteilnehmenden 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 Praxisrufümgesetzt. 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.

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

2 Vorgehensweise

2.1 Projektplan

Übersicht

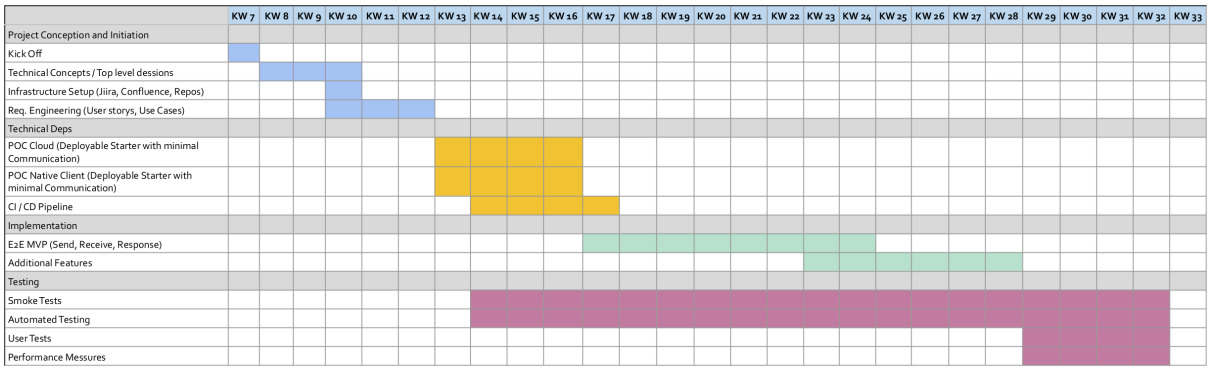


Abbildung 2.1: Projektplan

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 Umsetzten eines Proof Of Concepts. Dieser soll die gültigkeit der Resultate aus der ersten Phase verifizieren. Wenn nötig werden hier 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 beginnt in der dritten Phase (KW17-KW38). 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 zur Dritten 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 und auf Performance geprüft 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 Benachrichtungen 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	Setup Wizard für Konfiguration. Ein Administrator kann das System über eine Benutzeroberfläche konfigurieren.
M06	1:N Versenden Konfigurierbar. Es kann konfiguriert werden, welcher Client sich für welche Benachrichtigungen interessiert. Alle Clients erhalten nur für sie relevante Benachrichtigungen.
M07	Voice 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 verwendet die mobile Applikation von Praxisruf, um Benachrichtigungen zu versenden und empfangen. Im zweiten Bereich werden die Bedürfnisse von Praxisveranwotrlichen beschrieben. Diese Benutzergruppe ist dafür verantwortlich Praxisruf für Praxismitarbeitende zu konfigurieren. In einem dritten Bereich werden User Stories aus Sicht des Kunden festhalten, welche Rahmenbedingungen und Bedürfnisse des Auftraggebers festhalten. 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 notifikationen 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 einen 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

[2] Siehe 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 Verwendungsort 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 mit welchen anderen Geräten telefonieren können damit alle Mitarbeitenden 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

https://kotlinlang.org/lp/mobile/

+Jet Brains Infrastructure +We like Kotlin

-IoS Env. Needed to develop for Apple -Still has to develop separate API und UI Modules for Platforms

https://web.dev/progressive-web-apps/

+No need of Native Codebase +Perfect for Android -Eventually drawbacks because no entire API Access -PWAs on IOS suck

https://cordova.apache.org/

+ Popular Framework + Tons of plugins to access apis

-Still need to have a Mac for IoS development -Not a truly native app -¿ API Issues

https://nativescript.org/

+Provides a Workaround for nasty X-tools +Claims to be truly Native -Do we really trust it? (sorta new and passion project of a few people)

https://flutter.dev

-Why do you hate me?

SSimply Write Everything twice”

+Would definitely work

-Do most things twice -We don’t have time for that -Kunde wünscht ausdrücklich nur eine Codebasis für beide Clients.

https://stackshare.io/stackups/apache-cordova-vs-nativescript

https://nativescript.org/blog/build-nativescript-apps-remotely-from-windows-or-l

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 dabei nur über die Anbindung von Firebase Messaging (FCM) möglich.[2]

Um Mobile CLients über Benachrichtigungen zu informieren ist es möglich mit EventSources[3] Anfragen von der Serverseite an einen Client zu senden. Dies hat allerdings den Nachteil, dass eine Konstante HTTP Verbindung zwischen Client und Server bestehen muss.

Weiter besteht die Möglichkeit, über einen Messaging Broker Benachrichtigungen an den Mobile Client zu senden. Hier besteht allerdings dasselbe Problem. Damit Benachrichtigungen auf der Client Seite Empfangen werden können, muss eine Verbindung zum Messaging Broker bestehen. //TODO @Kevin: Any Idea how to cite this? I know it’s true, but citable source would be nice.

Sowohl die Option ËventSourcesäls auch die Option Message Broker"haben zudem die Einschränkung, dass sie es nicht ermöglichen Push-Benachrichtigungen anzuzeigen. Um dies auf iOS mit Nativescript zu ermöglichen müsste zusätzlich ein Firebase Messaging Service angebunden werden.

FCM ermöglicht es dabei selbst von einem Server Umfeld Benachrichtigungen an Endgeräte zu versenden.[4] Firebase 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 auschliessliche Firebase Messaging als Messaging Service verwendet.

4.3 Cloud Service

Cloud Services 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.[5] 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 T04[3] 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.[6]

[3] Siehe Kapitel 3

5 Konzept

5.1 Systemarchitektur

Abgrenzung

Dieses Kapitel gibt einen Übersicht über die Systemarchitektur als ganzes. Die Architektur beschränkt sich dabei auf die Anforderungen die innerhalb des Projektrahmens umgesetzt wurden. Komponenten für Teile die Out Of Scope 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 schliesslich von einem Mobile Client. Mit dem Mobile Client kann der Benutzer Benachrichtigungen an andere Mobile Clients senden. Welche Benachrichtigungen ein Mobile Client senden kann und an wen diese Benachrichtungen zugstellt werden, wird anhand der Konfiguration aus dem Cloud Service bestimmt.

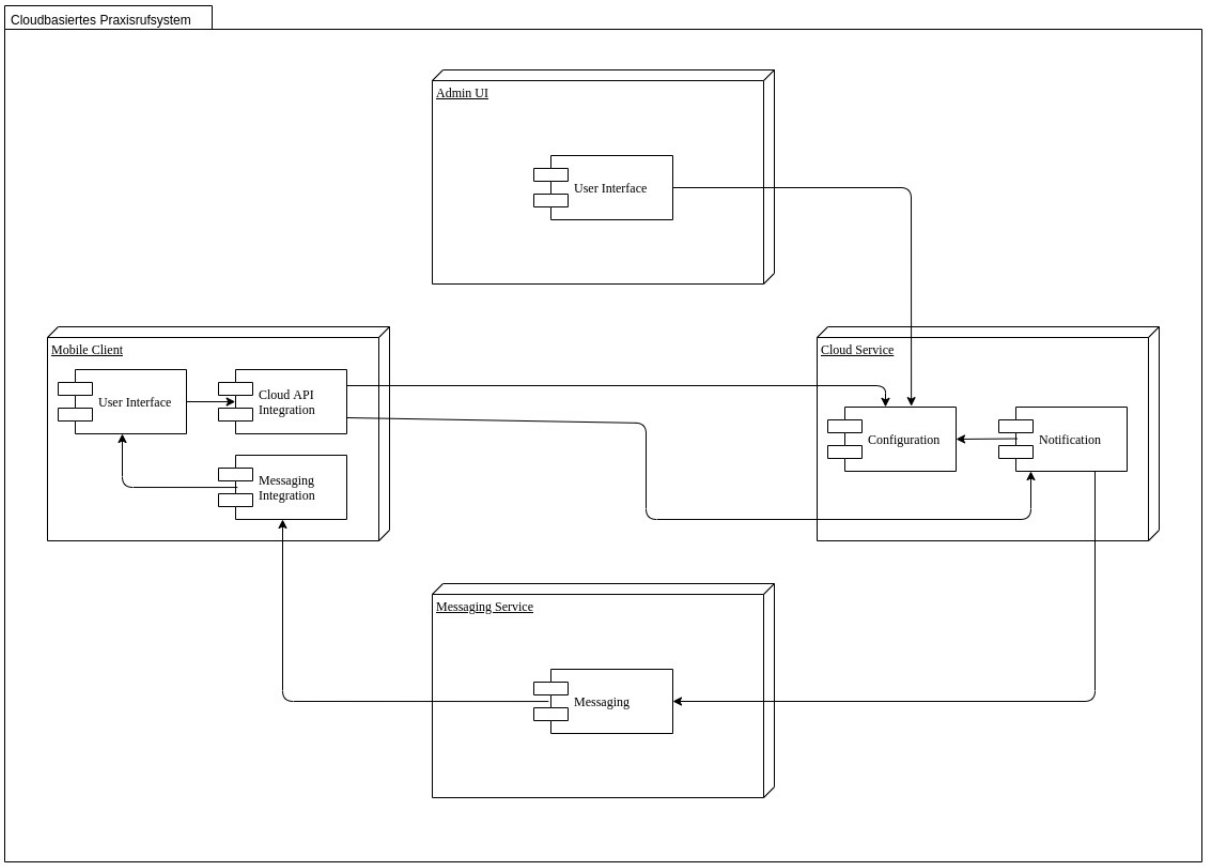


Abbildung 5.1: System

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 [7].

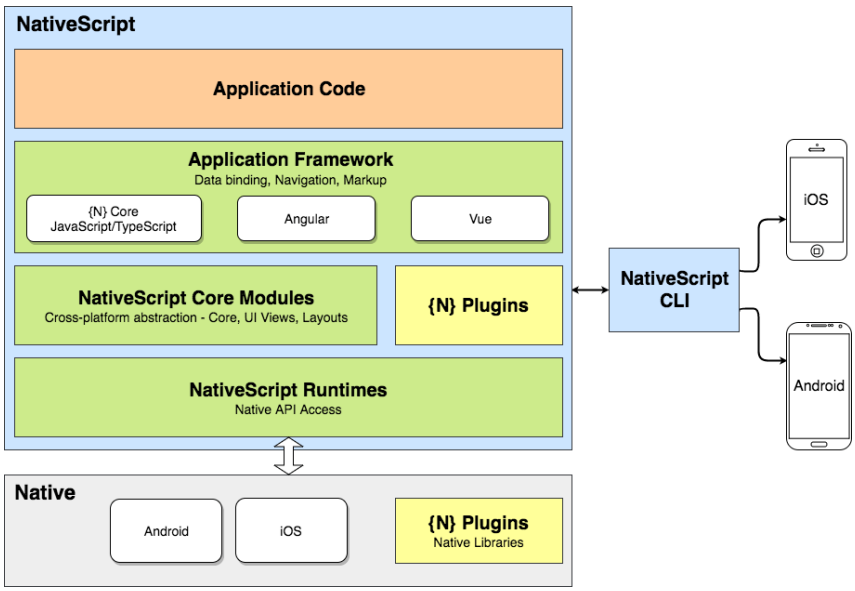


Abbildung 5.2: NativeScript-Overview
©OpenJS Foundation

Die Runtime agiert als Proxy zwischen Javascript und dem jeweiligen Ökosystem. Im Falle von IOS bedeutet dies u.A. das 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[8].

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>()` instanziiert 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

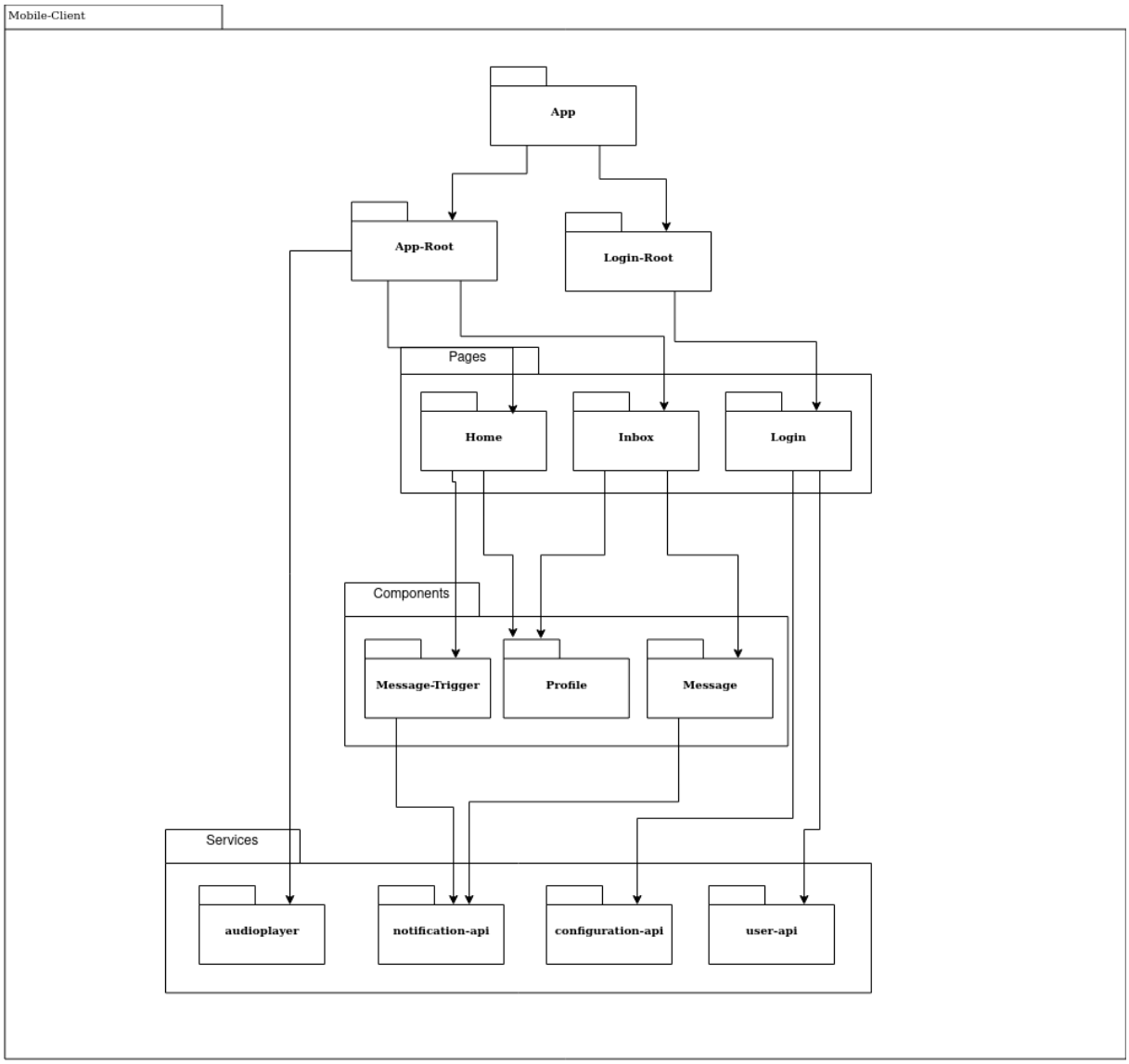


Abbildung 5.3: Mobile-Client Package Diagramm

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.

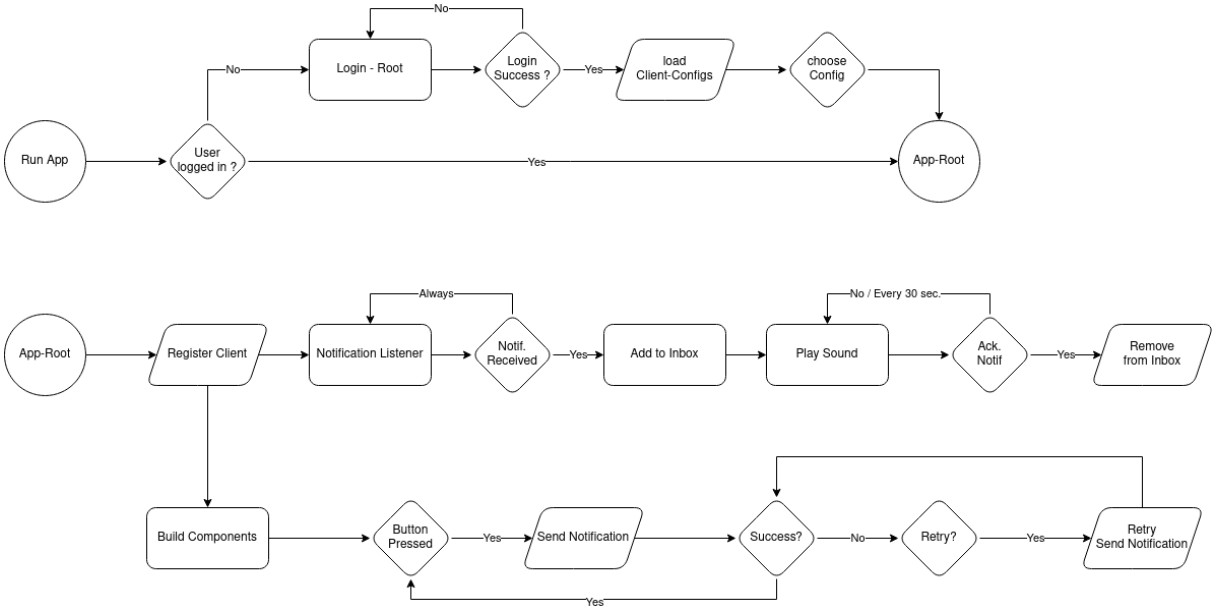


Abbildung 5.4: Mobile-Client Flow Chart

Das eigentliche *User Interface* besteht aus dem Homescreen, der es dem Benutzer erlaubt Nachrichten zu versenden, und der Inbox welche eingegangene Nachrichten anzeigt. Diese Ansicht ist erst nach erfolgreicher Authentifizierung erreichbar. Hat sich der Benutzer erfolgreich angemeldet, erhält er eine Auswahl der ihm zur verfügungstehenden 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 Cloudservices 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 auf Firebase aktiv. 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

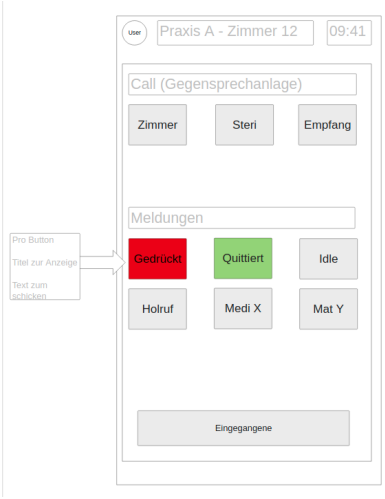


Abbildung 5.5: HomeScreen Mockup

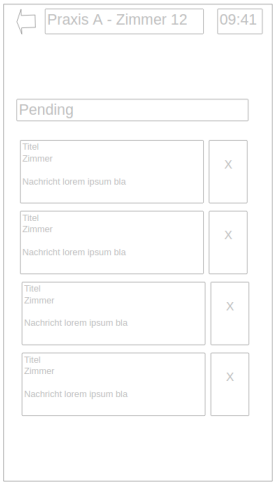


Abbildung 5.6: Inbox Mockup

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 Cloudservice eingegangen ist.

Die Gegensprechfunktion ist ebenfalls auf dem Homescreen angedacht. Hier sollten die Gesprächspartner direkt mit einem entsprechenden Button angewählt werden. Diese Funktionalität wie beschrieben in den Userstories U10 und U11 sind im Projektverlauf ausserhalb des Umsetzungsscopes gefallen und daher nicht weiter spezifiziert worden.

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

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

Das Icon könnte noch dynamisch mit dem jeweiligen Nachrichtentyp verknüpft werden. Solche wurden fachlich jedoch noch nicht definiert.

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 und die Information an welche Empfänger die Benachrichtigung versendet werden soll. Als Identifikation ist hier der technische Identifikator des Clients beim Messaging Service ausreichend.

Durch die klare Trennung der Verantwortungsbereiche der beiden Domains, 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 direkt komplizierter. Insbesondere Betrieb und Entwicklung wird aufwändiger, da diese Arbeiten nun über mehrere Applikationen verteilt sind. 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. 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 wird nach dem Prinzip von Onion Architecture aufgebaut.

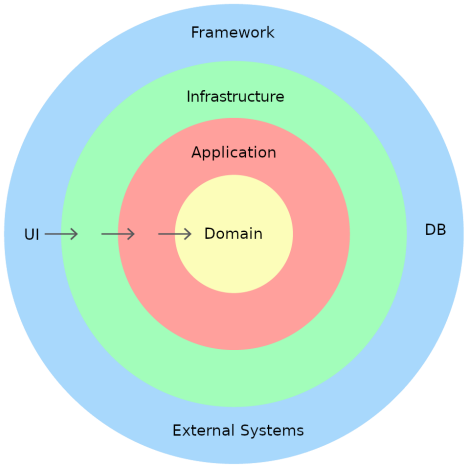


Abbildung 5.7: Clean Architecture

In Onion Architecture wird die Applikation in Layern von der Domäne im Zentrum bis hin zur Infrastruktur an den äusseren Enden definiert. Im Wesentlichen gibt es die folgenden Layers[9].

Domain Model Im Zentrum des Modells steht die Domain selbst. Der Domain Layer darf nur Dependencies auf sich selbst haben. Abhängigkeiten aus anderen Layers sind um jeden Preis zu vermeiden. Abhängigkeiten aus äusseren Layer auf den Domain Layer sind hingegen immer erlaubt.

Domain Services Bieten die fachliche Logik und Verhaltensweise des Domain Model Layers an.

Application Services Bildet die Brücke zwischen externer Infrastruktur und den DomainServices. Dies beinhaltet Repository Services und Rest Controllers.

Infrastructure Veranschaulicht die Infrastruktur ausserhalb des Systems. Dies beinhaltet unter anderem Datenbanken, Messaging Services und Applikationen die auf Web APIs welche der Applikation zugreifen.

Da auf die Aufteilung in Microservices für den Rahmen dieses Projektes verzichtet wird, muss sich diese Architektur innerhalb des Services wiederspiegeln. Dies soll durch die Aufteilung der Package Struktur geschehen.

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

Abbildung 5.8: Package Struktur Cloud Service

Im Zentrum steht das Package **domain**. Es beinhaltet alle Domänenobjekte und stellt damit alleine den Domain Layer dar.

Das Package **api** definiert Interfaces für sämtliche Domain Services. Es beinhaltet zudem Data Transition Objects (DTO), welche verwendet werden, um Daten nach aussen abzubilden. sowie Exceptions welche für den Cloud Service definiert werden.

Das Package **persistence** definiert Services welche für Interaktion mit der Datenbank verwendet werden und gehört zum Application Services Layer.

Das Package **service** implementiert die Services, welche im api Package vorgegeben sind und entspricht dem Domain Service Layer.

Das Package **web** definiert die REST Endpunkte, welche die Applikation nach aussen anbietet, sowie Clients die verwendet werden, um andere Systeme anzusprechen. Es gehört zum Application Services Layer.

Das Package **config** beinhaltet technische Konfiguration der Applikation.

5.3.2 Domäne Configuration

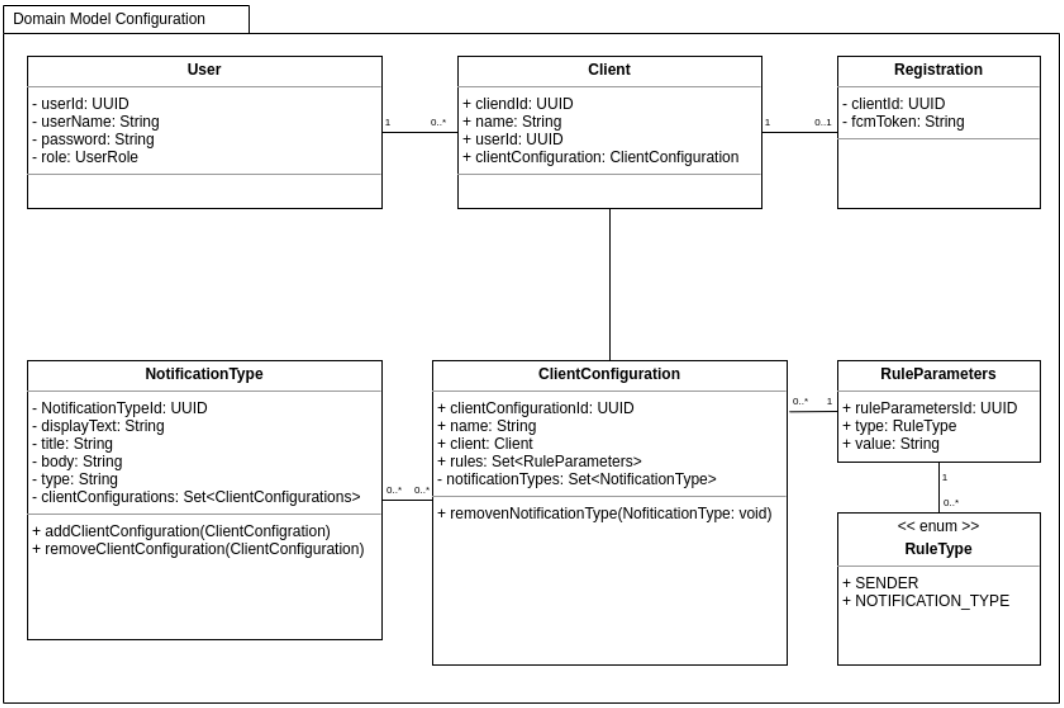


Abbildung 5.9: Domänenmodell Configuration

PraxisUser Ein Benutzer für das Praxisrufsystem. 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.

Client Clients repräsentieren ein Endgeräte, auf welchen die Mobile Client Applikation läuft.

Registration 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.

ClientConfiguration Stellt die Konfiguration eines Clients dar. Die ClientConiguration beinhaltet die Informationen, welche Benachrichtigung ein Client versenden kann und welche Benachrichtigungen er empfangen soll. Client und ClientConfiguration sind getrennt, damit Clients auch ohne Configuration erfasst werden können. So kann der Administrator bereits Geräte im System erfassen ohne Sie vollständig zu konfigurieren.

NotificationType Der NotificationType ist die Grundlage für Notifications die versendet werden. Der NotificationType definiert Informationen für den Inhalt von Benachrichtigungen die versendet werden. Er beinhaltet weiter Schlüssel für Texte die in einem Client angezeigt werden können.

RuleParameters RuleParameters werden pro ClientConfiguration erfasst. Die konfigurierten RuleParameters werden von der RulesEngine ausgewertet, um relevante Empfänger für eine Benachrichtigung zu finden.

RuleType Der RuleType bestimmt wie die Parameter für eine Regel ausgewertet werden. Für jeden Eintrag in der Tabelle RuleType muss es eine Implementierung des Interfaces RuleEvaluator geben, welche die RulesParameter Werte für den Regeltyp auswerten kann.

Die Configuration Domäne beinhaltet Services zum Lesen, Erstellen, Ändern und Löschen der Domänenobjekte. Diese Verwaltungsfunktionen werden über eine RESTful Web-API exponiert. [4] Der RegistrationService

[4] Siehe Kapitel 5.3.4

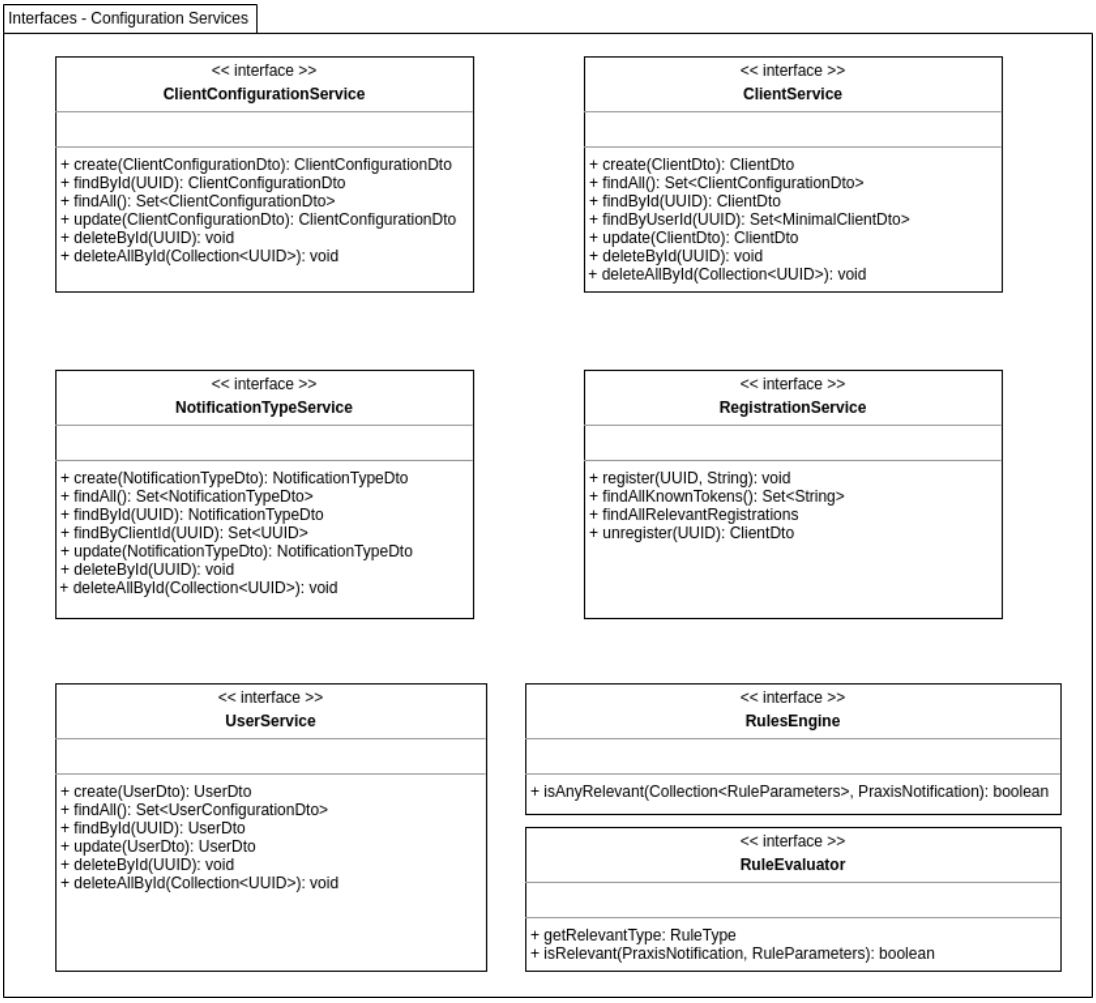


Abbildung 5.10: Klassendiagramm Configuration Service Interfaces

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 (RulesParameters) 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 RuleEvaluatoren delegiert. Umgesetzt wird dieses Konzept mit einem Strategy Pattern.[5] 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 bietet eine öffentliche Methode, über welche der RuleEvaluator für einen bestimmten Typ geladen werden

[5] Referece Gamma Design Patterns

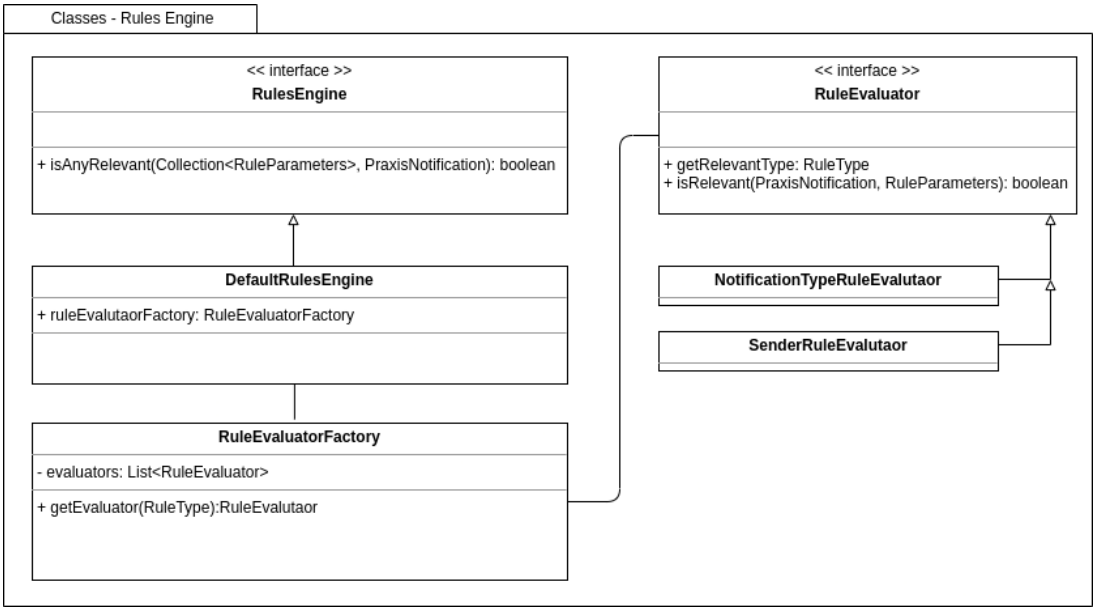


Abbildung 5.11: Klassendiagramm Rules Engine

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 RuleEvalutor 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 ahnand des RuleTypes kann nun über ein einfaches Lookup in der Map erfolgen. Werden in der Zukunft weitere RuleTypes unterstützt, reicht es die entsprechende RuleEvaluatorFactory zu implementieren. Der neue RuleEvaluator wird danach automatisch über dei RuleEvaluatorFactory verfügbar sein. Anpassungen an der RulesEngine oder der RuleEvaluatorFactory sind nicht nötig.

5.3.3 Domäne Notification

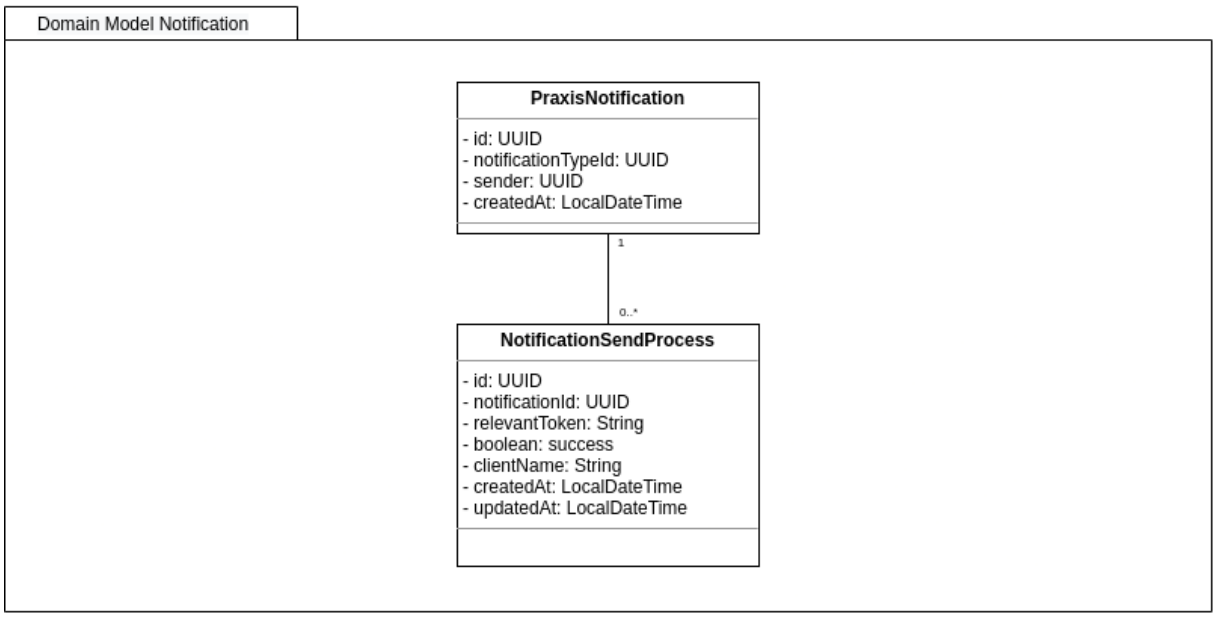


Abbildung 5.12: Domänenmodell Notification

PraxisNotification Jede Notification die der CloudService erhält wird bereits vor dem Versenden persistiert. Dies ermöglicht einzelne Benachrichtigungen die nicht versendet werden konnten zu wiederholen.

SendNotificationProcess Nach dem eine Benachrichtigung als PraxisNotification persistiert 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 ermöglicht es, eine Benachrichtigung nur für die Empfänger zu wiederholen, für die der Versand fehlgeschlagen ist.

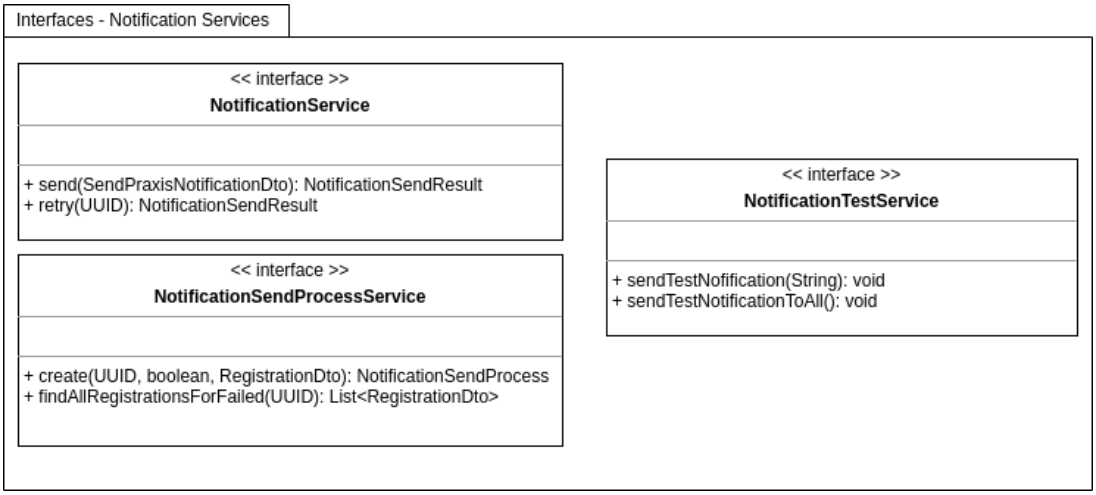


Abbildung 5.13: Klassendiagramm Notification Service Interfaces

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. Da die Notifikation und der Status des Versandprozesses beim ersten Versuch sie zu versenden persistiert wurde.

Der **NotificationSendProcessService** dient dazu Sendevorgänge zu erstellen und fehlgeschlagene Sendevorgänge zu finden. Dies ermöglicht es, eine Notifikation im Fehlerfall zu wiederholen.

Der **NotificationTestService** dient zu Test und Administrationszwecken. Er 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

Vorbedingung: Es wurde mindestens ein Client inklusive ClientConfiguration erfasst und dem Praxismitarbeiter zugewiesen.

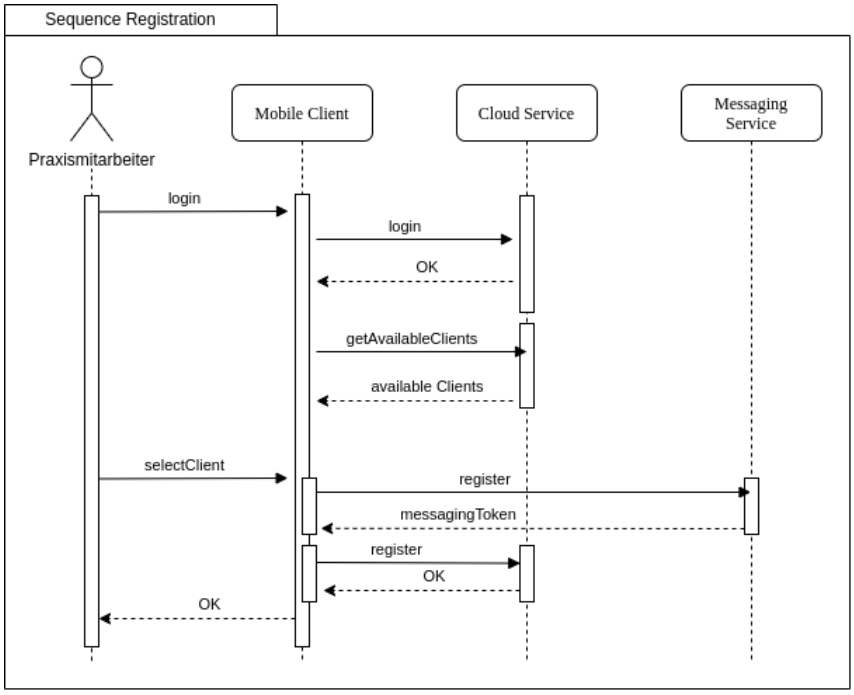


Abbildung 5.14: Ablauf Registration

In einem ersten Schritt muss sich der Praxismitarbeiter am Mobile Client anmelden. 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. Der Benutzer wählt darauf die gewünschte Konfiguration. Sobald die Konfiguration ausgewählt ist, 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 Messageing Service. Als Antwort erhält er ein eindeutiges Token, welches verwendet werden kann, um an diesem Client Nachrichten zu senden. Anschliessend registriert sich der Mobile Client mit dem erhaltenen Token und der ausgewählten Konfiguration beim Cloud Service. In diesem Zustand ist der Client bei Messaging Service und Cloud Service registriert und bereit Benachrichtigungen zu empfangen.

Benachrichtigung versenden und empfangen

Vorbedingung: Konfiguration und Registrierung gemäss Abbildung 5.14 ist für zwei Clients ist abgeschlossen. Konfiguration ist einer der Clients konfiguriert, Benachrichtigungen vom anderen Client zu empfangen.

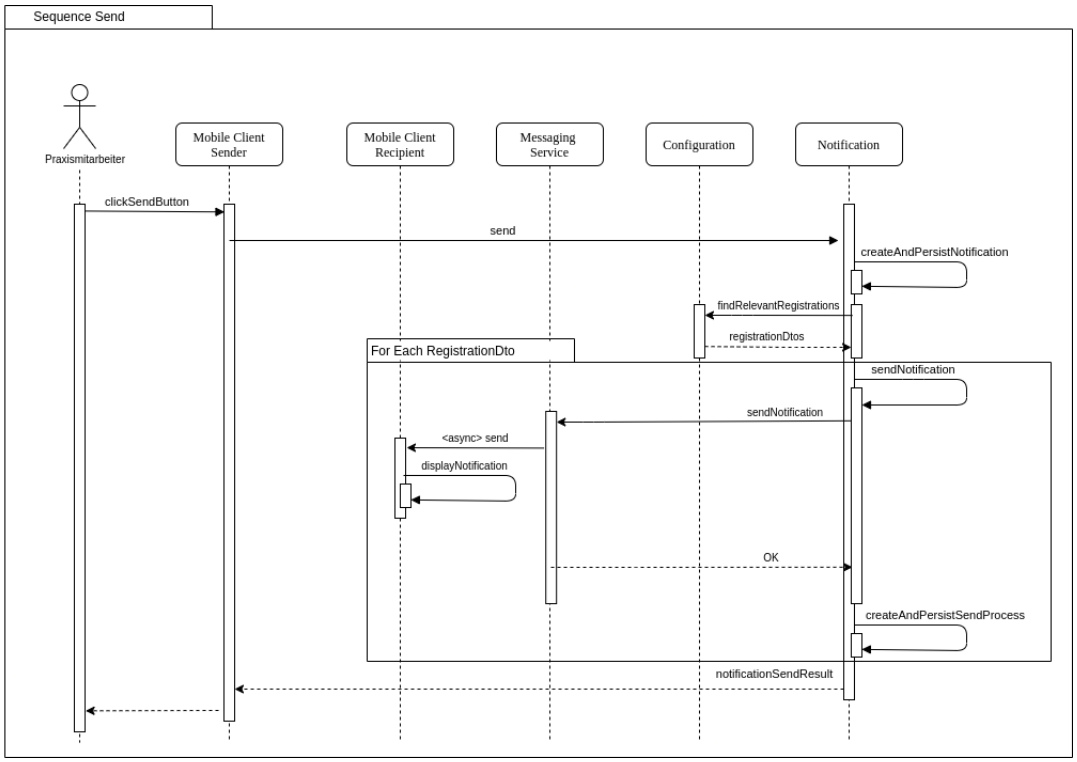


Abbildung 5.15: Ablauf Benachrichtigung Senden und Empfangen

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 Registrations relevanter Empfänger zurückgegeben. Sobald die Empfänger geladen sind, erstellt der NotificationController eine Benachrichtigung aus dem gegebenen NotificationType. Diese Benachrichtigung wird als PraxisNotification persistiert und anschliessned mit den geladenen Registrierungen an den Messaging Service gesendet, welcher die Zustellung an die entsprechenden Clients übernimmt. Nach dem Übermitteln an den Messaging Service wird pro Empfänger ein NotificationSendProzess erstellt der den Status für diesen Empfänger dokumentiert. Der NotificationController sender darauf eine Antwort zurück. Diese Antwort beinhaltet die technische Id der versendeten Benachrichtigung und das Resultat ob die Benachrichtigung an alle Empfänger versendet werden konnte. Ist dies nicht der Fall, wird der Retry-Process auf Client Seite gestartet.

Auf Empfängerseite wird, sobald die Benachrichtigung verarbeitet und angezeigt sobald die Zustellung durch den Messaging Service erfolgt ist. Diese Zustellung erfolgt asynchron. Kann der Sender nur informiert werden, wenn die Zustellung an den Messaging Service fehlgeschlagen ist.

Benachrichtigung wiederholen

Vorbedingung: Konfiguration und Registrierung gemäss Abbildung 5.14 ist für zwei Clients ist abgeschlossen. Konfiguration ist einer der Clients konfiguriert, Benachrichtigungen vom anderen Client zu empfangen. Es wurde eine Benachrichtigung gemäss Abbildung 5.15 versendet. Dabei ist das Versenden an mindestens einen Client 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. NotificationController durchsucht die NotificationSendProcess Tabelle nach der gegebenen id und filtert auf fehlgeschlagene. Anschliessend wird der SendProcess anhand der Tokens in dieser NotificationSendProcess Instanzen wiederholt.

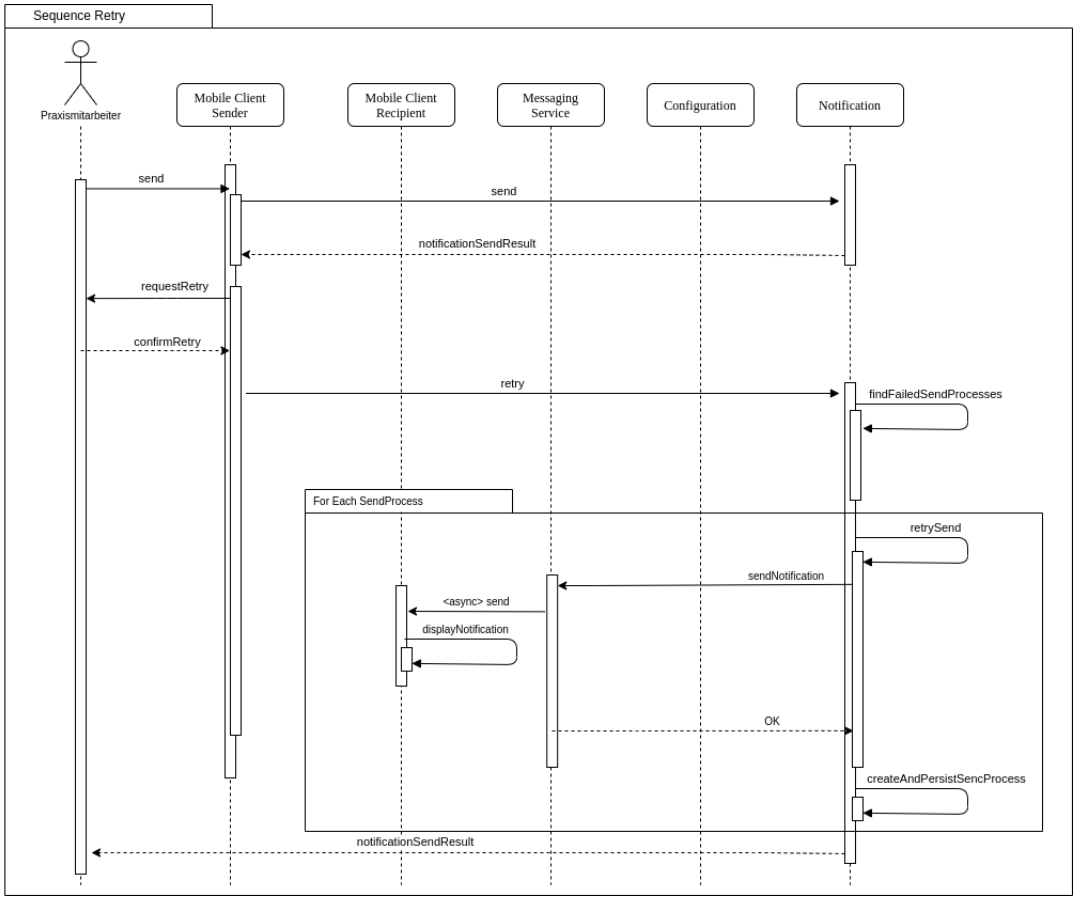


Abbildung 5.16: Ablauf Benachrichtigung Wiederholen

5.3.5 API

Um die Verwaltung der Konfigurationen zu ermöglichen, bietet der Cloud Service eine REST-API an über die Konfigurationsobjekte verwaltet werden können. Der Cloud Service bietet Verwaltungs APIs für die Folgenden Domänenobjekte:

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

Für jedes dieser Domänenobjekte ist ein dedizierter Endpoint definiert, der unter dem Subpfad /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 wird analog zum Domänenservice für Registrations[6] folgende API angeboten:

Aktion	**HTTP**	**Pfad**	**Body**	**Response**
Registrierung aktualisiern	POST	/api/registrations/	ClientId, Token	-
Registrierung entfernen	DELETE	/api/registrations/	ClientId	-
Relevante Registrierung finden	POST	/api/registrations/tokens	NotificationDto	[RegistrationDto]

Benachrichtigungen

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

Aktion	**HTTP**	**Pfad**	**Body**	**Response**
Registrierung aktualisiern	POST	/api/registrations/	ClientId, Token	-
Registrierung entfernen	DELETE	/api/registrations/	ClientId	-
Relevante Registrierung finden	POST	/api/registrations/tokens	NotificationDto	[RegistrationDto]

[6] Siehe Kapitel 5.3.2

Authentifizierung

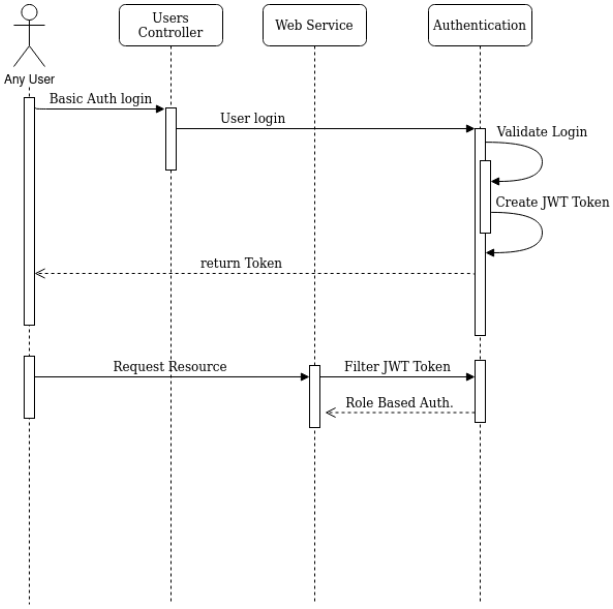


Abbildung 5.17: Authentifizierung-Sequenz

5.4 Admin UI

Für die übergeordnete Administrations-Oberfläche wurde auf Vorschlag des Kunden ein React-Admin Projekt erstellt.

5.4.1 Framework Grundlagen

React Admin ist ein Framework, spezialisiert für CRUD oberflächen. 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 [10].

5.4.2 Anwendung

Die konfigurierbaren Elemente werden jeweils in einem eigenen Component verwaltet. React Admin unterstützt bereits Relationen zwischen diesen Komponenten.

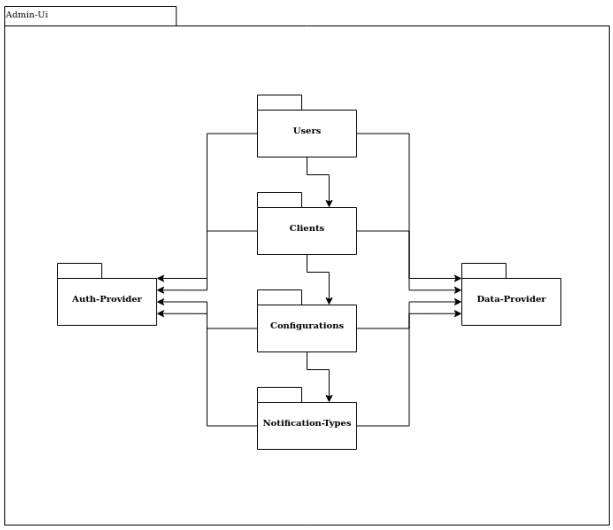


Abbildung 5.18: Admin-Ui Package Diagramm

Die Requests werden von jeweils vom 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 versendenmit 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 empfangenmit 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 Foregroundünd ”Backgroundmit 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. Kommunikation mit Cloud Service muss funktionieren. Kommunikation mit Messaging Service muss funktionieren.

T04 - AWS Platform

Der Cloud Service muss auf AWS deployed werden. Die Kommunikation zwischen Mobile Client und Cloud Service muss funktionieren. Die Kommunikation mit Messaging Service muss funktionieren.

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 können. Aus dem Mobile Client muss der Benutzer über einen Butten eine Benachrichtigung versenden können. Das Versenden dieser Benachrichtigung erfolgt an den Cloud Service. Im Rahmen des Proof Of Concept wird eine Benachrichtigung immer an den Sender zurückgesendet. Dabei ist aber wichtig, dass die Benachrichtigung nicht direkt als Antwort auf die Versenden-Anfrage geschickt wird. Stattdessen muss das Versenden der Benachrichtigung aus dem Cloud Service über den Message Service erfolgen. .

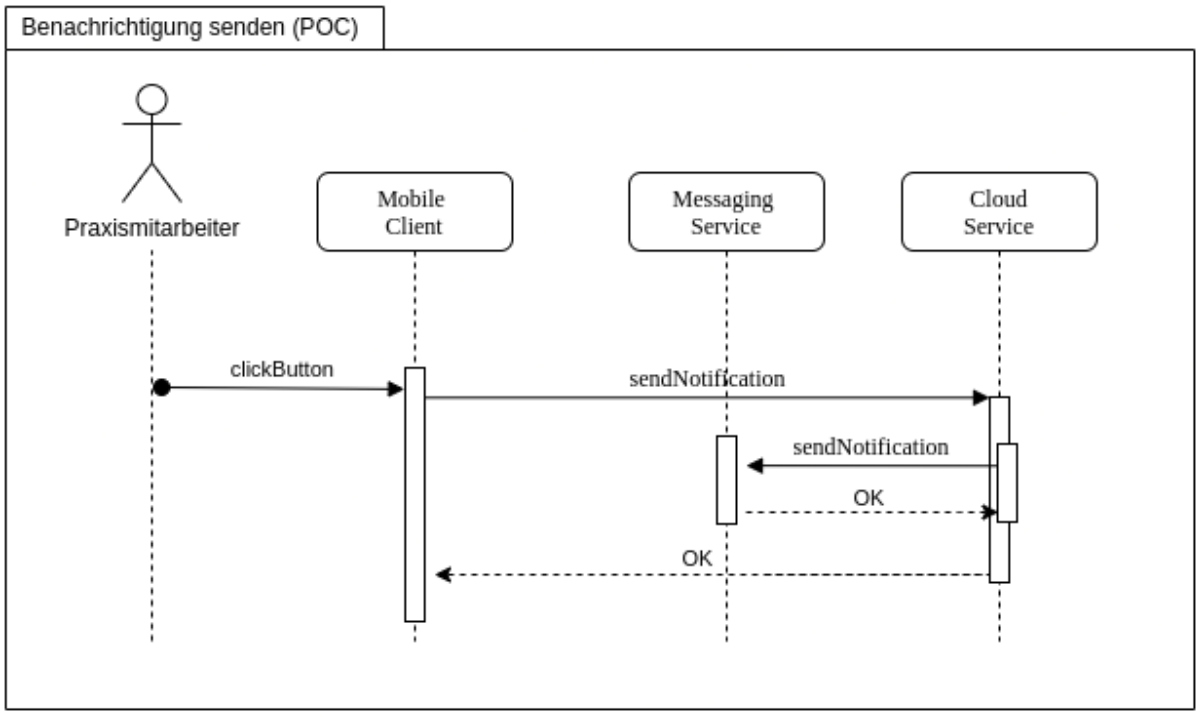


Abbildung 5.19: Proof Of Concept - Benachrichtigung versenden

Wurde eine Benachrichtigung über den Message Service an den Mobile Client versendet, muss diese vom Mobile Client empfangen werden. Als Reaktion auf den Empfang der Benachrichtigung, muss der Inhalt der Benachrichtigung im Mobile Client angezeigt werden. Zudem muss eine Push Benachrichtigung auf dem Host Gerät erfolgen.

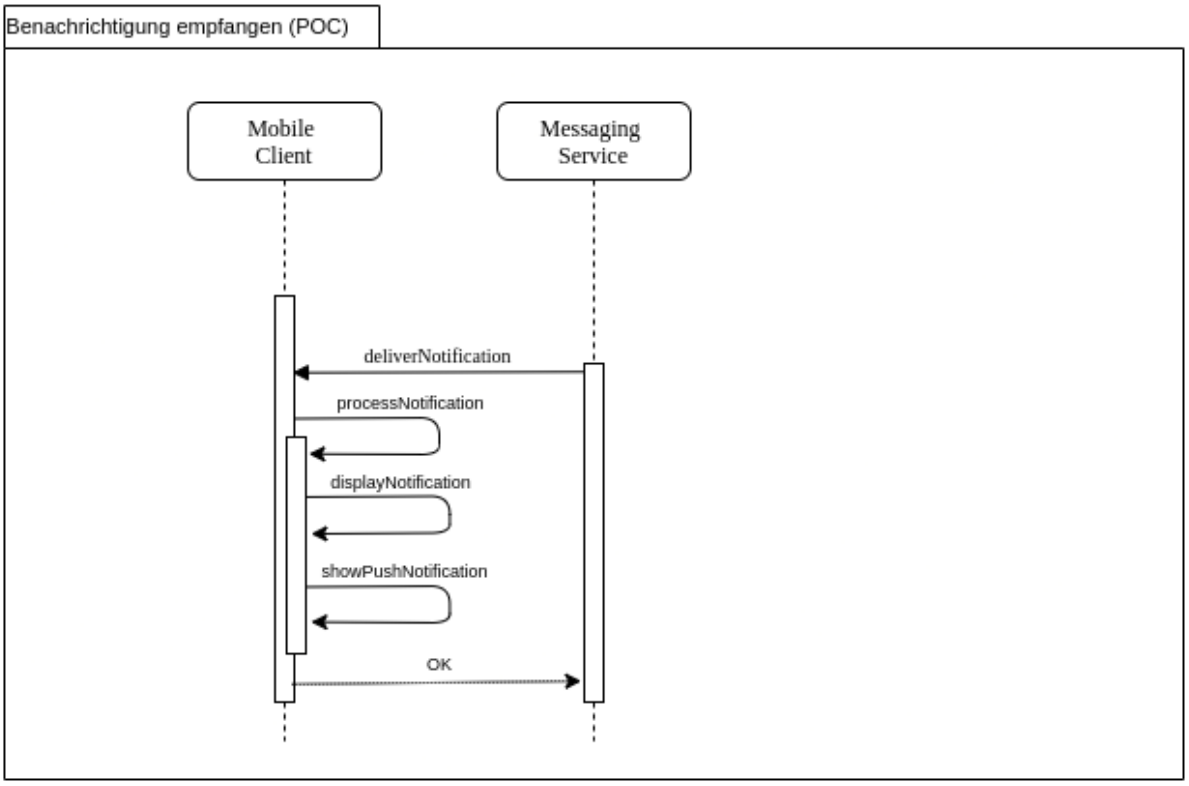


Abbildung 5.20: Proof Of Concept - Benachrichtigung empfangen

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 als Firebase 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 kann[7].

Im Rahmen des Projektes wurden damit die Meilensteine M01 bis M06[8] erreicht. Umgesetzt wurden die Meilensteine mit den folgenden User StoriesSiehe Kapitel 3 inklusive aller dazu definierten Features und Szenarien[9]:

- 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 and IOS
- T04 - Betrieb mit AWS

Die Meilensteine M06 bis M10 konnten im Rahmen dieses Projektes nicht umgesetzt werden. Dementsprechend wurden keine Anforderungen dazu umgesetzt. Dies betrifft die User Stories U08 bis U16.

- 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

[7] Siehe Anhang D
[8] Siehe Kapitel 2.2
[9] Siehe 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 entnehmen[10].

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 Benutzername 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 Home Seite weitergeleitet.

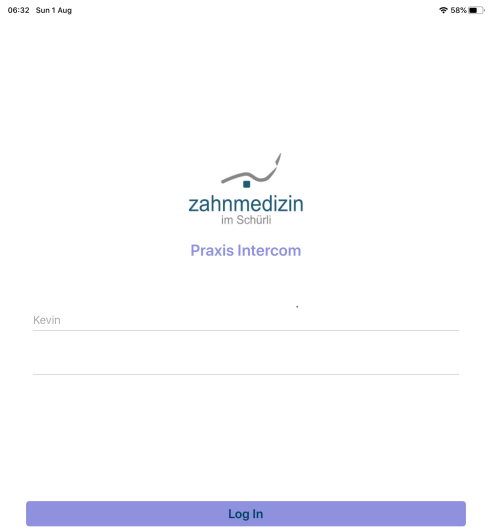


Abbildung 6.1: Login

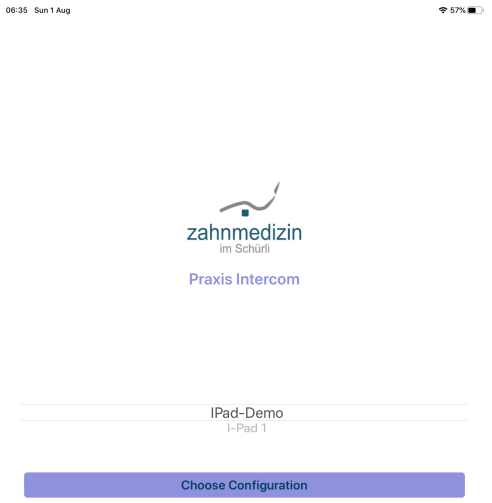


Abbildung 6.2: Konfiguration

[10] Siehe 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.

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

Abbildung 6.3: Home

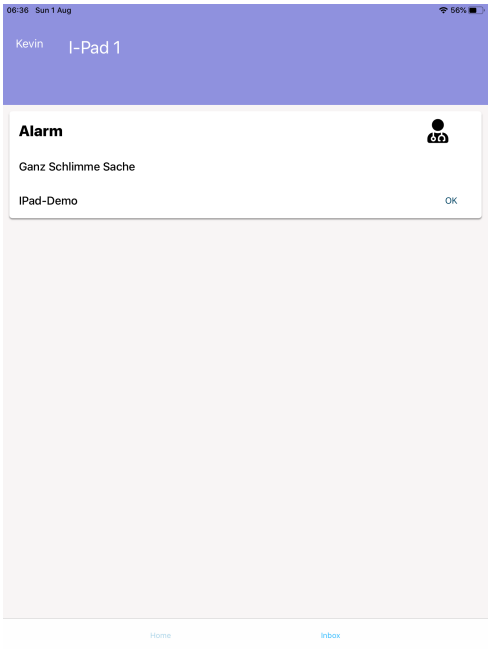


Abbildung 6.4: Retry

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 X Sekunden das Audiosignal. Diese Quittierung erfolgt dabei 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

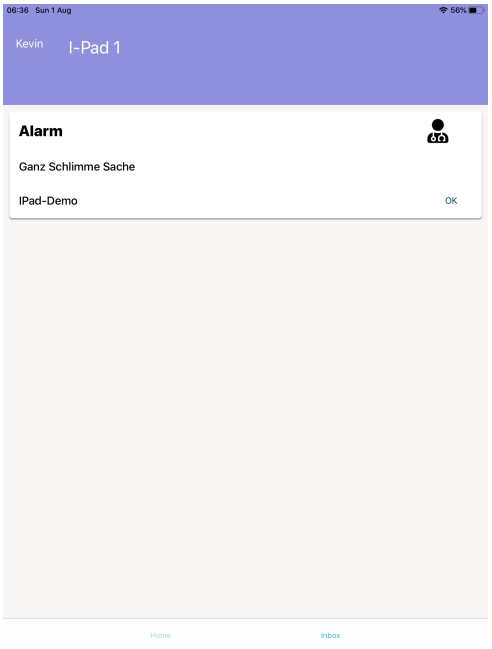


Abbildung 6.6: Push Benachrichtigung

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, um sie gemäss Konfiguration über den angebundenen Message Service an andere Mobile Clients zu weiterzuleiten.

Um diese Aufgaben zu erfüllen, bietet der Cloud Service eine REST API 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 abgelegt[11]. Dies Beinhaltet die verfügbaren Endpunkte und das Model welches für Input und Ouput Werte dieser API dienen.

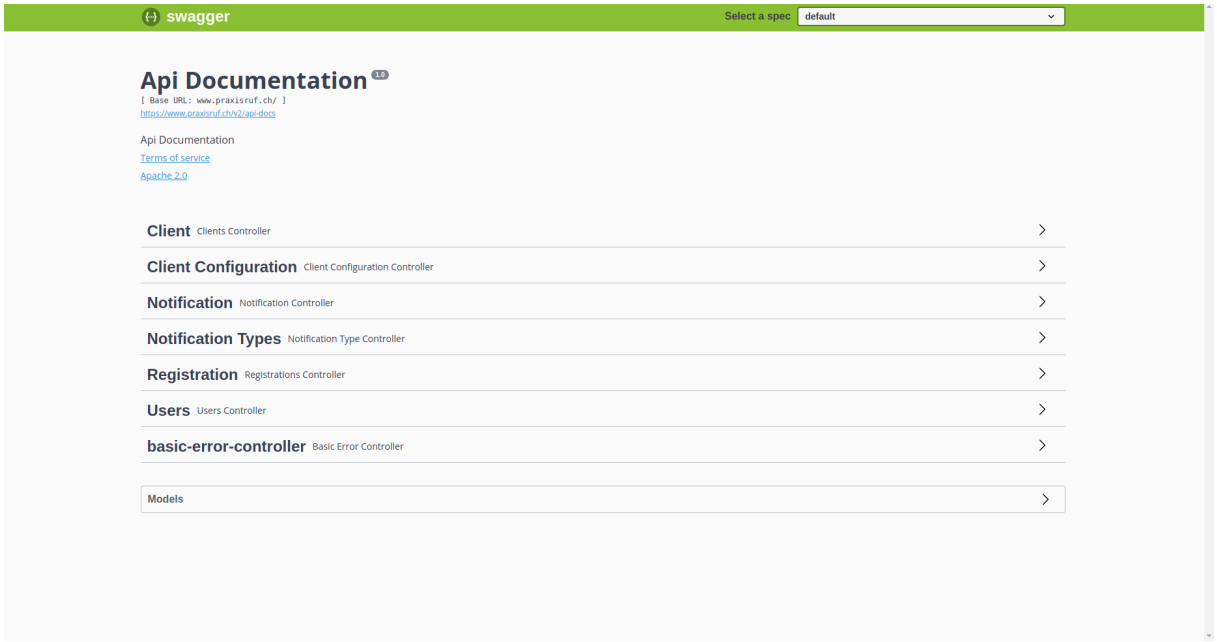


Abbildung 6.7: Swagger UI

[11] Siehe Anhang B

6.1.3 Admin UI

Mit dem Admin UI bietet das Praxisrufsystem dem Praxisverantwortlichen die Möglichkeit die Konfiguration des Systems zu verwalten. Da nur der Praxisverantworliche das Admin UI verwenden darf, ist die Benutzeroberfläche durch ein Login geschützt. Die entsprechenden Anmeldeinformationen müssen bei Installation des Cloud Services vom Betreiber manuell konfiguriert werden.

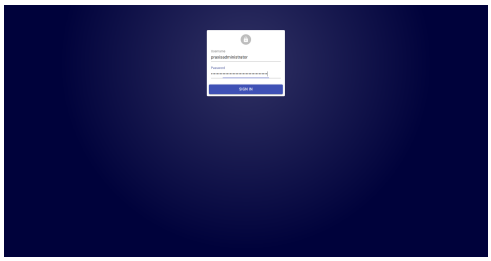


Abbildung 6.8: Login

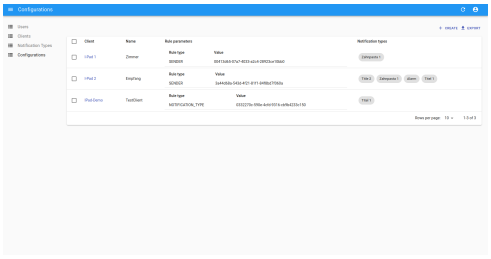


Abbildung 6.9: Configuration Overview

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.

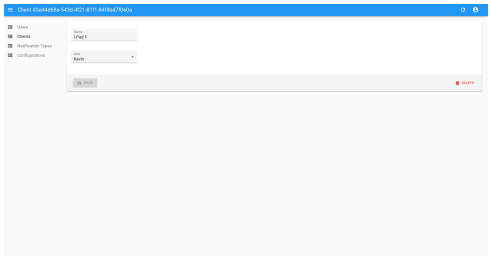


Abbildung 6.10: Login

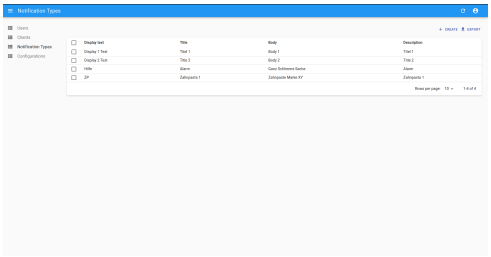


Abbildung 6.11: Configuration Overview

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.[12]

[12] Siehe Anhang C

6.2 Tests

Benutzertests

Testablauf

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

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.

Ale Ergebnis wird erwartet, dass die Benachrichtigung auf dem physischen Ipad ankommt und angezeigt wird. Schritte x bis y werden mit minimiertem Mobile Client wiederholt. Dabei wird erwartet dass eine Push Benachrichtigung angezeigt wird.

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

Feedback vom Benutzer

Als Feedback vom Kunden haben wir zwei weitere Funktionswünsche erhalten:

- 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. Diese Anforderungen konnten auch noch umgesetzt werden. Der dritte Wunsch des Kunden, die Quittierung von Benachrichtungen an den Sender weitergeleitet wird konnte aus zeitlichen Gründen nicht mehr umgesetzt werden.

Testplan

Aus den nach den Benutzertests vervollständigten Features und Szenarien wurde der finale Testplan für das Umgesetzte System definiert. Alle Szenarien sind detailliert im Kapitel 3.2 beschrieben. Jedes Szenario definiert die erwarteten Vorbedingungen, einen Testschritt und die zu verifizierenden Ergebnisse. Diese Szenarien dienen als Grundlage für den Testplan. Dabei bildet jedes Szenario einen Testfälle. Diese Test Cases wurden auf dem umgesetzten System ausgeführt. Die folgende Tabelle zeigt die Resultate der Testfälle.

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 and 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 Appliaktion 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 Aufsetzten 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 dienen[13].

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

[13] Siehe Anhang D

Projekt. Dabei ist es wichtig, dass auch Betriebsinfrastruktur, Skalierbarkait 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, der es Erlaubt 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 Herausforderungen[14] 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 Gruppengespräche 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 verhältnissmässig 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 einfachere Wartung. Es erschwert allerdings die Anbindung von Betriebssystemfunktionen und Zugriff auf Gerätehardeware 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.

[14] Siehe Kapitel 6

Literaturverzeichnis

[1] D. Jossen, $21FS_IMVS01 : CloudbasiertesPraxisrufsystem$, 2020.

[2] farfromrefuge. (). NativeScript Push-Benachrichtigungen, Adresse: `https://market.nativescript.org/plugins/nativescript-push/`.

[3] M. D. Network. (). EventSource, Adresse: `https://developer.mozilla.org/en-US/docs/Web/API/EventSource`.

[4] G. Developers. (). Your server environment and FCM, Adresse: `https://firebase.google.com/docs/cloud-messaging/server`.

[5] V. Inc. (). Why Spring? Adresse: `https://spring.io/why-spring`.

[6] J. Villa. (). Spring Boot with AWS, Adresse: `https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html`.

[7] O. Foundation. (). How NativeScript Works, Adresse: `https://v7.docs.nativescript.org/core-concepts/technical-overview`.

[8] ——, (). What is iOS Runtime for NativeScript? Adresse: `https://v7.docs.nativescript.org/core-concepts/ios-runtime/overview`.

[9] S. Odean. (). Software Architecture — The Onion Architecture, Adresse: `https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec`.

[10] Marmelab. (). react-admin, Adresse: `https://marmelab.com/react-admin/Readme.html`.

Abbildungsverzeichnis

A Aufgabenstellung

Fachhochschule Nordwestschweiz

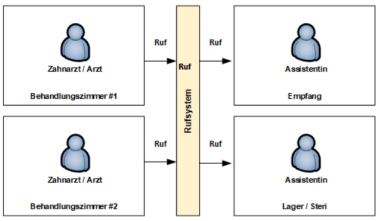
App
Web

21FS_IMVS01: Cloudbasiertes Praxisrufsystem

			Priorität 1	Priorität 2
Betreuer:	Daniel Jossen	**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 Gegensprechfunktion zu integrieren. Ein durchgeführte Marktanalyse hat gezeigt, dass die meisten auf dem Markt kommerziell erhältlichen Rufsysteme auf proprietären Standards beruhen und ein veraltetes Bussystem oder analoge Funktechnologie zur Signalübermittlung einsetzen. Weiter können diese Systeme nicht in ein TCP/IP-Netzwerk integriert werden und über eine API extern angesteuert werden.



Ziel der Arbeit

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

- 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 Kommunikaion)
- 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 Mutlicast Übertragung der Daten ermöglicht. Über eine offene Systemarchitektur müssen die Kommunikationsbuttons in der App frei konfiguriert und parametrisiert werden können.

Technologien/Fachliche Schwerpunkte/Referenzen

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

Abbildung A.1: Aufgabenstellung

B Quellcodeverwaltung

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 einsehbar. (Referenz https://github.com/IP5-Cloudbasiertes-Praxisrufsystem). Berechtigungen können bei Joshua Villing oder Kevin Zellweger angefordert werden.

C Benutzerhandbuch

Mobile Client

Anmeldung und Konfiguration

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

Benachrichtigung empfangen

1. In Mobile Client anmelden.
2. In Navigationsleiste (unten) ”Homeäuswählen.
3. Button mit dem gewünschten Titel antippen.
4. Die Benachrichtigung wird automatisch versendet.

Benachrichtigung Auflisten und Quittieren

1. In Mobile Client anmelden.
2. In Navigationsleiste (unten) Ïnboxäuswählen.
3. In dieser Ansicht sehen Sie alle empfangenen noch nicht quittierten Benachrichtigungen.
4. Zum Quittieren einer Benachrichtigung kann diese angetippt werden.

Abmeldung

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

Admin UI

Anmeldung

1. Admin UI im Browser öffnen
2. Anmeldedaten eingeben und bestätigen. Die Anmeldedaten erhalten Sie nach Installation des Systems vom Betreiber.

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 Createüm 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 löschenüm einen Eintrag zu löschen.

Praxisruf konfigurieren

1. Melden Sie sich im Admin UI an.
2. Erfassen Sie in der Kategorie Üserpro Praxiszimmer einen Benutzer.
3. Erfassen Sie in der Kategorie Clientpro Gerät und Zimmer einen Eintrag und weisen Sie ihn dem Benutzer des entsprechenden Zimmers zu.

4. Erfassen Sie in der Kategorie Notification Typesälle Benachrichtigungen die Sie zur Verfügung stellen möchten.
5. Erfassen Sie in der Kategorie Configurationspro 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.

D Installationsanleitung

Mobile Client

TODO
?? Cloud Service url configuration
?? FCM Integration configuration

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 Anbdingun 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[15] um den nötigen AWS Amplify Service zu installieren.
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 Ënvironment Variables"
   (c) Erstellen Sie eine neue Variable mit dem Namen REACT_APP_BACKEND_BASE_URI. Setzten 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 [16] 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.[17]
   (b) Wählen Sie unter Plattform "Javaünd die dazugehörigen Standardeinstellungen.
   (c) Wählen Sie unter Application Code SSample Application".
3. Erstellen sie mit AWS RDS eine Datenbank für die Cloud Service Applikation
   (a) In Beanstalk Console
   (b) Folgen Sie der offiziellen Anleitung[18] um eine Relationale Datenbank an das Beanstalk Environment anzubinden.
   (c) Wählen Sie als Datenbank Engine postgresïn 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.

[15] https://docs.aws.amazon.com/amplify/latest/userguide/getting-started.html
[16] https://docs.aws.amazon.com/amplify/latest/userguide/custom-domains.html
[17] https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.CreateApp.html
[18] https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html

4. Definieren Sie die nötigen Umgebungsvariablen für die Cloud Service Applikation:
   (a) Folgen Sie der offiziellen Anleitung [19] um die nötigen Umgebungsvariablen zu setzen:
   (b) Name: FCM_CREDENTIALS, Wert: Firebase Credentials mit Base 64 Encoded[20]
   (c) Name: SPRING_PROFILES_ACTIVE, Wert: aws.
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 Pipelineäus.
   (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 Üse 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 Pipelineäus.
   (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 Eingabem.
7. Stellen Sie sicher, dass die installierete Cloud Service Applikation über HTTPS erreichbar ist. Dies ist für die Kommunication mit Mobile Client Instanzen zwingend notwendig.
   (a) Folgen Sie dazu der offiziellen Anleitung[21].
   (b) Am Cloud Service sind dabei keine Änderungen nötig.
8. Erstellen Sie einen Adminstrator Account für das Admin UI.
   (a) Richten Sie den Datenbankzugriff auf die in Schritt 3 erstellte Datenbank mit den Datenbanktool ihrer Wahl ein.[22]
   (b) 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

[19] https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environments-cfg-softwaresettings.html
[20] Siehe Installationsanleitung Firebase Messaging
[21] https://aws.amazon.com/premiumsupport/knowledge-center/elastic-beanstalk-https-configuration/
[22] https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER$_{C}onnectToPostgreSQLInstance.html$

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:	Benachrichtitung 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:	Benachrichtitung 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 klick, 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 emfpangen
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 is geöffnet
When:	Benutzer ist nicht angemeldet
Then:	Benutzer wird zum Login aufgefordert
Scenario S11:	Startbildschirm wenn angemeldet
Given:	Mobile Client is 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 .....