

Cloudbasiertes Praxisrufsystem

IP 5

7. August 2021



Abbildung 0.1: Titlebild

Studenten	Joshua Villing, Kevin Zellweger
Fachbetreuer	Daniel Jossen
Auftraggeberin	Daniel Jossen
Studiengang	Informatik
Hochschule	Hochschule für Technik

Zusammenfassung

Das Abstract ist eine Art Zusammenfassung des ganzen Dokuments. Es gibt einen Einblick in die Aufgabenstellung, wie diese umgesetzt wurde und welches Ergebnis erreicht wurde. Aus diesem Grund wird das Abstract immer ganz am Schluss der Arbeit verfasst. Es besteht aus einem zusammengehörenden Absatz und umfasst ungefähr 10 bis 20 Zeilen. Formeln, Referenzen oder andere Unterbrechungen haben im Text nichts zu suchen. Direkt unter dem Abstract folgt eine Liste von drei bis vier Stichworten/Keywords. Diese werden in alphabetischer Reihenfolge aufgelistet und beschreiben das Themengebiet der Arbeit.

Keywords: Anleitung, LaTeX, Thesis, Vorlage

Management Summary siehe PF-IK.

Inhaltsverzeichnis

1	Einleitung	1
2	Vorgehensweise	3
2.1	Stakeholder	3
2.2	Projektplan	4
2.3	Organisation	6
3	Anforderungen	7
3.1	User Stories	7
3.2	Features	10
4	Evaluation Technologien	14
4.1	Mobile Client Evaluation	14
4.2	Cloud Service	14
4.3	Betrieb und Plattform	14
5	Konzept	15
5.1	Systemarchitektur	15
5.2	Mobile Client	17
5.3	Cloud Service	23
5.4	Admin UI	38
5.5	Proof Of Concept	39
6	Umsetzung	42
6.1	Resultate	42
6.2	Tests	48
6.3	Herausforderungen	50
6.4	Lessons Learned	52
7	Schluss	53
	Abbildungsverzeichnis	55

8	Anhang	56
8.1	Aufgabenstellung	56
8.2	Benutzerhandbuch	57
8.3	Betriebshandbuch	57
8.4	Entwicklerdokumentation	57
8.5	Ehrlichkeitserklärung	57

1 Einleitung

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. Eine 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 Gegensprechfunktion (1:1 oder 1:m Kommunikation)
- Auf der App können beliebige Buttons konfiguriert werden, die anschließend auf den anderen Tablets einen Alarm oder eine Meldung (Text- oder Sprachmeldung) generieren
- Textnachrichten können als Sprachnachricht (Text to Speech) ausgegeben werden
- Verschlüsselte Übertragung aller Meldungen zwischen den einzelnen Stationen
- Integration der Lösung in den Zahnarztbehandlungsstuhl über einen Raspberry PI
- Offene API für Integration in Praxisadministrationssystem

Problemstellung

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

Methodik

Das Projekt Cloudbasiertes Praxisrufsystem soll in vier Phasen umgesetzt werden. In der ersten Phase wird die Organisation innerhalb des Projekt teams geklärt. Es werden die Top Level Anforderungen besprochen und priorisiert. Dazu werden oberflächliche Konzepte erarbeitet. In der zweiten Phase sollen Technologien evaluiert werden die verwendet werden um die wichtigsten Anforderungen umzusetzen. Es wird ein Proof Of Concept implementiert, der beweist, dass die technischen Voraussetzungen gegeben sind, diese Anforderungen umzusetzen. In der dritten Phase wird auf Basis des Proof Of Concept das Praxisrufsystem umgesetzt. Die umzusetzenden Anforderungen sind dabei in einem iterativen Verfahren zusammen mit dem Kunden zu spezifizieren. Die aktuelle Lage wird in Regelmässigen Absätzen mit dem Kunden besprochen und die nächsten Schritte werden geklärt.

Konzepterläuterung

Im nachfolgenden Hauptteil wird die Arbeit im Detail vorgestellt. Der Hauptteil gliedert sich in fünf Teile. Zuerst werden im Kapitel 2 Vorgehensweise der Projektplan und die Organisation im vorgestellt. Anschliessend zeigt das Kapitel 3 Anforderungen, welche Anforderungen mit dem Kunden erarbeitet und umgesetzt wurden. Das Kapitel 4 Evaluation Technologien stellt die Recherchen vor die gemacht wurden um die für die Anforderungen geeigneten Technologien zu identifizieren. Das Kapitel 5 Konzept stellt schliesslich das detaillierte technische Konzept vor, das dem umgesetzten System zugrunde liegt. Die Resultate der Arbeit werden dann im Kapitel 6 Umsetzung vorgestellt.

2 Vorgehensweise

2.1 Stakeholder

Am Projekt IP5 Cloudbasiertes Praxisrufsystem sind folgende drei Stakeholder beteiligt.

Prof. Daniel Jossen

- Rolle: Auftraggeber und Betreuer
- Kontakt: daniel.jossen@fhnw.ch

Joshua Villing

- Rolle: Student
- Kontakt: joshua.villing@students.fhnw.ch

Kevin Zellweger

- Rolle: Student
- Kontakt: kevin.zellweger@students.fhnw.ch

2.2 Projektplan

Übersicht

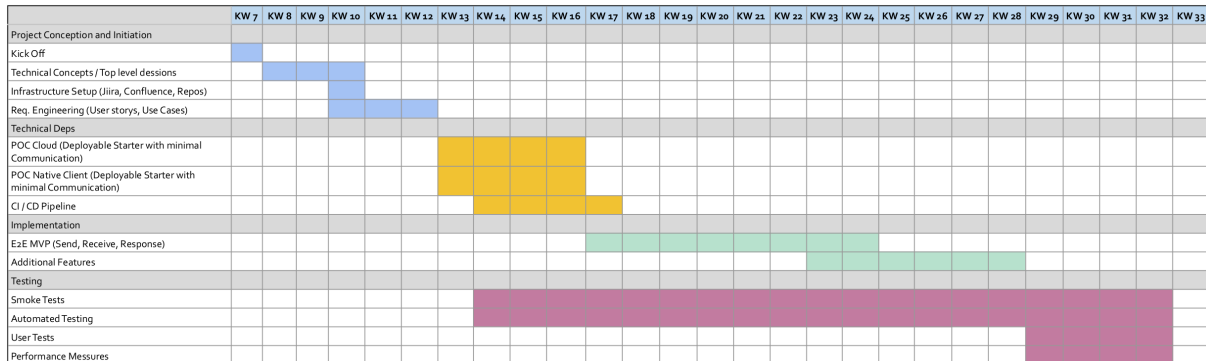


Abbildung 2.1: Projektplan

Das Projekt Cloudbasiertes Praxisrufsystem soll in vier Phasen umgesetzt werden. Wobei sich diese Phasen teilweise überschneiden.

In der ersten Phase (KW7-KW13) soll die Infrastruktur die für das Projekt benötigt wird aufgebaut werden. Es sollen Top Level Konzepte erfasst werden, wie das System funktionieren und kommunizieren soll. Es sollen Technologien evaluiert werden, die zur Umsetzung verwendet werden können. Es sollen die Prioritäten und Wichtigsten Ziele des Projektes geklärt werden. Diese Prioritäten und Ziele werden in Milestones festgehalten. (Siehe Kapitel XY).

In der zweiten Phase (KW13-KW17) soll ein Proof Of Concept erstellt werden. Der die Recherchen und Konzepte aus Phase 1 verifiziert. Wenn nötig werden hier Anpassungen an den gewählten Technologien gemacht. Die Architektur und das Konzept werden weiter verfeinert. Anforderungen werden konkretisiert.

In der dritten Phase (KW17-KW38) wird das Projekt schliesslich umgesetzt. Zur Umsetzung gehört dabei vorzu die nächsten Anforderungen die Umzusetzen sind zu besprechen und die Konzepte dafür zu erweitern. Als erster Teil soll dabei ein Minimal Viable Produkt erstellt werden. Dieser MVP muss mit einem Backend kommunizieren können und Benachrichtigungen versenden und empfangen können.

Die vierte Phase (KW14-KW32) läuft parallel zur Dritten und beschäftigt sich mit dem Testing. 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.

Milestones

In der Anfangsphase des Projektes wurden folgende Milestones definiert:

Id	Beschreibung
M01	Proof Of Concept - Setup. Ein Mobile Client kann auf IOS installiert werden und mit einem Backendservice kommunizieren.
M02	Proof Of Concept - Messaging. Ein Mobile Client kann Benachrichtigungen empfangen und Push-Benachrichtigungen anzeigen.
M03	Versenden mit Registration. Mobile Clients können sich beim Praxisrufsystem registrieren und Benachrichtigungen empfangen. Alle registrierten Clients erhalten alle Benachrichtigungen.
M04	Benachrichtigungen konfigurierbar. Der Mobile Client lädt seine Konfiguration vom Backend und zeigt dynamisch konfigurierte Buttons an über die Benachrichtigungen versendet werden.
M05	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.

2.3 Organisation

Kommunikation

Das Projekt IP5 Cloudbasiertes Praxisrufsystem wurde im FS21 gestartet. Die Organisation und Kommunikation des Projektes mussten dementsprechend für die Einschränkungen wegen Corona angepasst werden. Um sicherzustellen, dass die Kommunikation über die gesamte Projektdauer funktionieren kann, haben wir uns deshalb von Anfang an entschieden die Kommunikation über Remote- und Online Tools zu organisieren. Für Besprechungen und Planungen wurde Microsoft Teams gewählt. Die entsprechende Infrastruktur wurde von der FHNW zur Verfügung gestellt.

Dokumentation

Der Bericht wurde mit LaTeX und zusammen mit dem Quellcode verwaltet. Kurze Besprechungen, Notizen und interne Dokumentation erfolgten über ein geteiltes One Note Notizbuch.

Sämtliche Diagramme, Mockups und Skizzen wurden direkt in den Tools verwaltet, die zur Erstellung gebraucht wurden. Zum Schluss wurden alle für den Bericht relevanten Darstellungen exportiert und in den Bericht integriert.

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.

- IP5-praxis-mobile-client
- IP5-praxis-cloud-service
- IP5-praxis-admin-ui
- IP5-praxis-documentation

Tools und Werkzeuge

- draw.io
- moqus.com
- Visual Studio Code
- IntelliJ
- Git
- github.com

3 Anforderungen

Die im Rahmen des Projektes umzusetzenden Anforderungen wurden während des Projektes iterativ zusammen mit dem Kunden erarbeitet. Alle Anforderungen werden zuerst aus Fachlicher Sicht mit User Stories festgehalten, die ein konkretes Bedürfnis der Benutzer beschreiben. Weiter werden User Stories aus Sicht des Kunden festgehalten, welche Rahmenbedingungen und Bedürfnisse des Auftraggebers festhalten. Aufgrund der User Stories werden anschliessend Features definiert, welche konkrete Szenarien und die erwarteten Ergebnisse an definieren.

3.1 User Stories

Praxismitarbeiter

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

Praxisverantwortlicher

Id	Anforderung	Features
U12	Als Praxisverantwortlicher möchte ich mehrere Geräte verwalten können, damit das Praxisrufsystem in mehreren Zimmern gleichzeitig genutzt werden kann.	F0x
U13	Als Praxisverantwortlicher möchte ich definieren können welches Gerät, welche Anfragen versenden kann, damit jedes Gerät Verwendungsort optimiert ist.	F0x
U14	Als Praxisverantwortlicher möchte ich die Konfiguration des Praxisrufsystems zentral verwalten können, damit das Praxisrufsystem für die Anwender optimiert werden kann.	F0x
U15	Als Praxisverantwortlicher möchte ich definieren können, welche Geräte mit welchen anderen Geräten telefonieren können damit meinen Mitarbeitern das Arbeiten erleichtere.	F0x
U16	Als Praxisverantwortlicher möchte ich definieren können, welche Benachrichtigung über einen physischen Knopf am Behandlungsstuhl versendet wird damit der Knopf für den Mitarbeiter optimiert ist.	F0x

Auftraggeber

Id	Anforderung	Features
T01	Als Auftraggeber möchte ich, dass das Praxisrufsystem über iPads bedient werden kann, damit ich von bestehender Infrastruktur profitieren kann.	F0x
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.	F0x
T03	Als Auftraggeber möchte ich, dass die Codebasis für das Praxisrufsystem für Android und IOS verwendet werden kann, damit ich die Weiterentwicklung optimieren kann.	F0x
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.	F0x

3.2 Features

F01 - Benachrichtigungen Versenden

Scenario S01 Benachrichtigung versenden

Given: Benutzer ist vollständig angemeldet
And: Mindestens 1 Empfänger ist konfiguriert
When: Praxismitarbeiter tippt auf einen Benachrichtigungs-Button
Then: Benachrichtigung wird an den zentralen Cloud Service gesendet
And: Benachrichtigung wird an alle Mobile Clients versendet
die sich für diese Benachrichtigung subscribed haben weitergeleitet
And: Praxismitarbeiter erhält optische Rückmeldung, dass Benachrichtigung versendet wurde

Scenario S02 Keine Empfänger konfiguriert

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

F02 - Benachrichtigungen Empfangen

Scenario S03 Empfangen

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

F03 - Fehlgeschlagene Benachrichtigungen

Scenario S04 Fehler Rückmeldung

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

Scenario: 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: Cancel Retry

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

F04 - Über Benachrichtigungen Notifizieren

Scenario S05 Foreground

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

Scenario S06 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 S07 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 S08 Startbildschirm wenn nicht angemeldet

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

Scenario S09 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 S10 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 möchte

Scenario S11 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 S12 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 S13 Logout

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

F06 - Konfigurationsverwaltung

Scenario S14Login

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 S15Anmelden 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 S16Konfiguration 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.

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 -i 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 Cloud Service

<https://aws.amazon.com/>

<https://spring.io/projects/spring-boot>

Konfig der Clients könnte sich als No-SQL anbieten.

Config muss nur gelesen und an den Client geschickt oder abgespeichert werden

<https://www.mongodb.com/>

4.3 Betrieb und Platform

AWS ist MUSS

5 Konzept

5.1 Systemarchitektur

Abgrenzung

Dieses Kapitel gibt einen Überblick 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 cloud-basierte 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 Benachrichtigungen zugestellt werden, wird anhand der Konfiguration aus dem Cloud Service bestimmt.

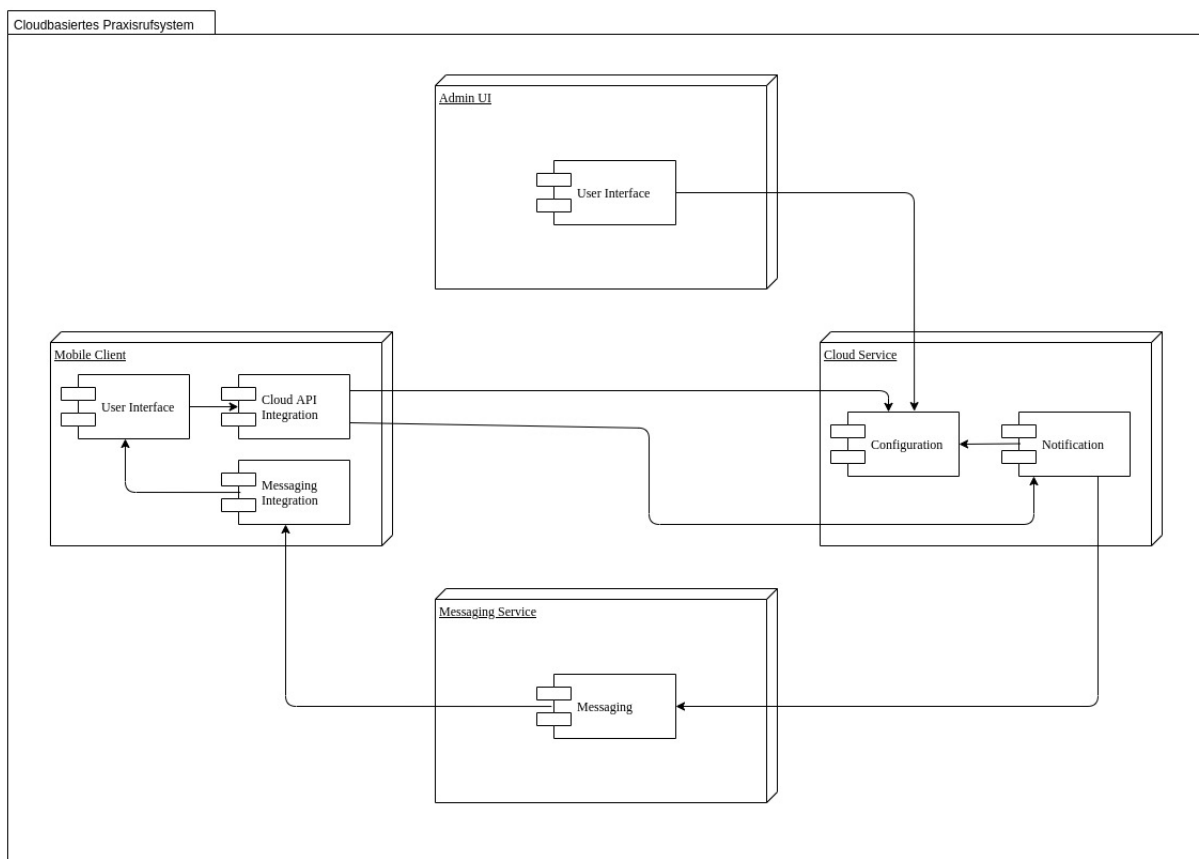


Abbildung 5.1: System

Mobile Client

Damit der Praxismitarbeiter wie in den Anforderungen U01 und U02 beschrieben Benachrichtigungen versenden und empfangen kann, benötigt das System einen Client, welcher dem Praxismitarbeiter diese Operationen ermöglicht. Die technischen Anforderungen T01, T02 und T03 setzen voraus, dass dieser Client über die Mobil Geräte (IPads und Android Tablets) bedient werden kann.

- T01
- T02
- T03
- U01
- U02
- U03
- U04
- U05
- U06
- U07

Cloud Service

Der Cloud Service 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.

- U01
- U02
- U03
- U06
- U07

Messaging Service

Der Messaging Service ist für die Zustellung von Benachrichtigungen verantwortlich.

- U01
- U02
- U03
- U04
- U06

Admin UI

Damit der Praxisverantwortliche die Konfiguration des Praxisrufsystems verwalten kann ist es notwendig, dass alle Lese und Schreib Zugriffe auf die Konfiguration über eine Zentrale Stelle gehen, auf die der Praxisverantwortliche Zugriff hat.

- U12
- U13
- U14
- T04

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 [1].



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[2].

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



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
9         auto auto" horizontalAlignment="center">
10
11       </GridLayout>
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

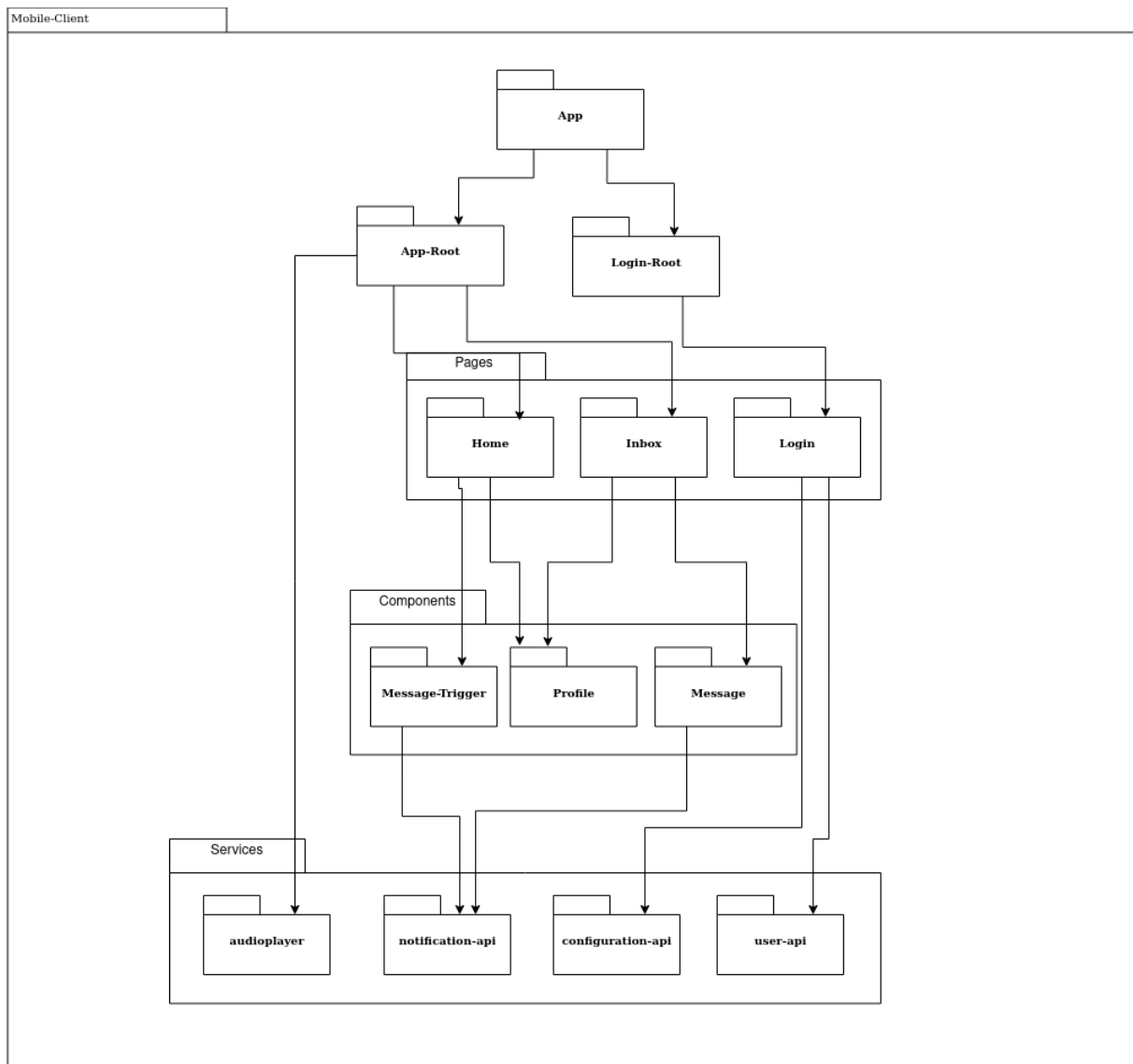


Abbildung 5.3: Mobile-Client Package Diagramm

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 Umsetzungssscopes 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

Überblick

Der Cloud Service wird in die zwei Domänen Notification und Configuration aufgeteilt. Dabei ist die Domäne Notification für das Versenden von Benachrichtigungen zuständig. Die Domäne Configuration ist für die Verwaltung der Konfigurationen und Subscriptions verantwortlich. Die Domäne Notification benötigt zum Versenden von Benachrichtigungen Informationen, die aus der Domäne Configuration stammen. Grundsätzlich braucht sie dazu aber nur ein kleines Subset dieser Informationen. Sobald identifiziert ist, an wen eine Benachrichtigung gesendet werden soll und wie dieser Client erreicht werden kann, funktioniert das Versenden der Benachrichtigung komplett unabhängig von der Domäne Configuration.

Da die beiden Domänen nicht eng gekoppelt sind, wäre es denkbar die Domänenspezifische Logik in zwei separate Microservices aufzuteilen. 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 in als eine einzelne Spring Boot Applikation umgesetzt. Innerhalb dieser Applikation sollen die Domänen Configuration und Notification 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.

Service Architektur

Der Cloud Service wird nach dem Prinzip von Onion Architecture / Clean Architecture aufgebaut.

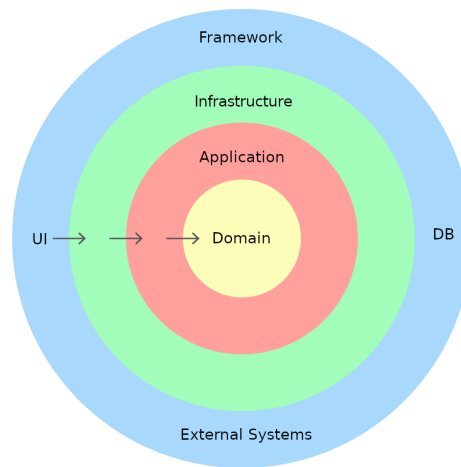


Abbildung 5.7: Clean Architecture

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[3].

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.

Package Struktur

Der Cloud Service definiert die folgenden top level Packages:

```
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 weiter Data Transition Objects (DTOs), welche verwendet werden um Daten ausserhalb der Domain (UI oder andere Domain) abzubilden. Es beinhaltet zudem Exceptions welche für den Cloud Service definiert werden.

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

Das Package **service** beinhaltet die Services welche das Verhalten der Applikation modellieren und entspricht dem Domain Service Layer.

Das Package **web** beinhaltet die Controller welche REST Endpoints anbieten, sowie Clients die verwendet werden um andere REST Endpoints zu konsumieren. Es gehört zum Application Services Layer.

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

5.3.2 Domänenmodell

Wie in Kapitel 4.3.1 beschrieben, teilt sich der Cloud Service in die zwei Domänenbereiche Notification und Configuration auf. Im Folgenden wird detailliert beschrieben, wie diese Domänen aufgebaut sind.

Domäne Configuration

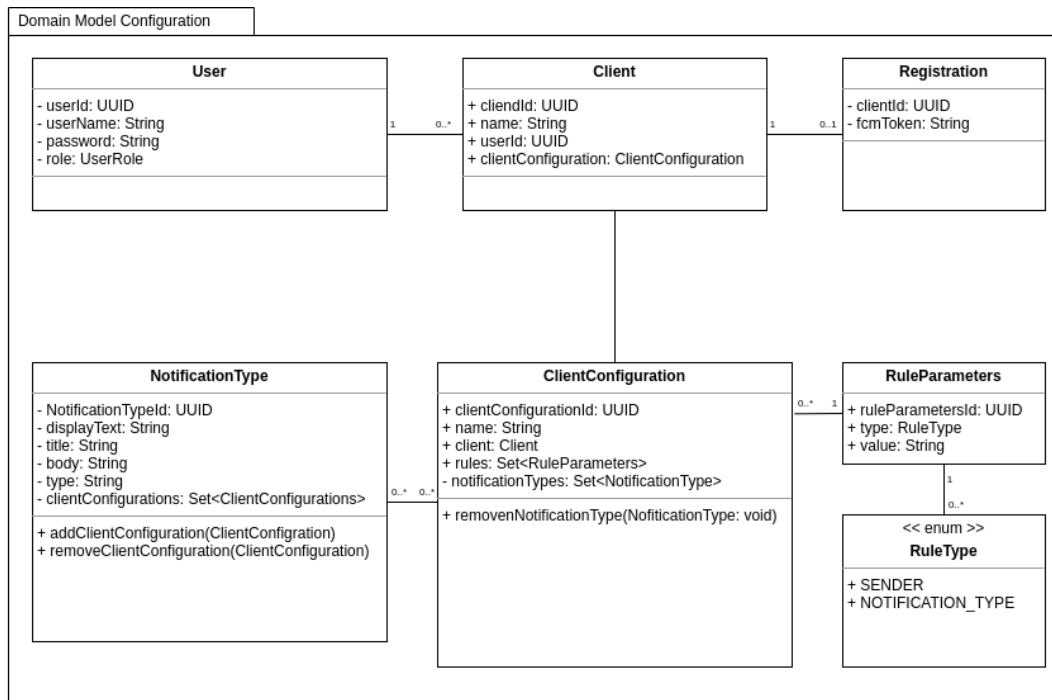


Abbildung 5.9: Domänenmodell Configuration

PraxisUser Ein Benutzer für das Praxisrufsystem. Jeder Benutzer kann entweder die Rolle Admin oder User haben. Admin ist für Admin UI (Praxisverantwortlicher). User ist für Mobile Client (Praxismitarbeiter) Nutzer.

Client Ein Client repräsentiert ein Gerät auf welchem der Mobile Client läuft.

Registration Die Registrierung eines Clients beinhaltet die Informationen die vom Messaging Service verwendet werden können um diesem spezifischen Client eine Benachrichtigung zu stellen. Ein Client kann nie mehrere Registrierungen haben.

ClientConfiguration Stellt die Konfiguration eines Clients dar. Die Beziehung von Client zu ClientConfiguration ist immer 1:1. Client und ClientConfiguration sind getrennt, damit Clients auch ohne Configuration erfasst werden können. So kann der Administrator die Geräte die er hat von Anfang an erfassen aber erst nach und nach die Konfigurationen dazu erstellen.

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 sind die parameterisierten Werte für eine Regel. Sie werden pro ClientConfiguration erfasst. Sie werden von der RulesEngine ausgewertet um zu identifizieren ob ein Client sich für eine gegebene Benachrichtigung interessiert.

RuleType Der RuleType bestimmt wie die Parameter für eine Regel (siehe RuleParameters) 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.

Configuration Services



Abbildung 5.10: Klassendiagramm Configuration Service Interfaces

Abbildung 4.6 zeigt die Interfaces für Services welche es in der Configuration Domäne gibt. ClientConfigurationService, ClientService, NotificationTypeService, RegistrationService und UserService bieten Create, Read, Update und Delete Methoden für das jeweilige Domänenobjekt an. Diese Methoden sind zu Verwaltungszwecken über eine REST-API ansprechbar. Siehe Kapitel xy (API).

Der RegistrationService bietet dabei eine Ausnahme. Der RegistrationService bietet Methoden um Registrierungen zu erfassen und wieder zu löschen. Er bietet zudem die Möglichkeit anhand einer gegebenen Notification zu identifizieren, welche der registrierten Clients sich für diese Notification interessieren. Diese Auswertungen werden mithilfe der RulesEngine gemacht. Die RulesEngine bewertet die erfassten RulesParameter mithilfe der RuleEvaluators und findet alle relevanten tokens. Siehe Kapitel Rules Engine.

Rules Engine

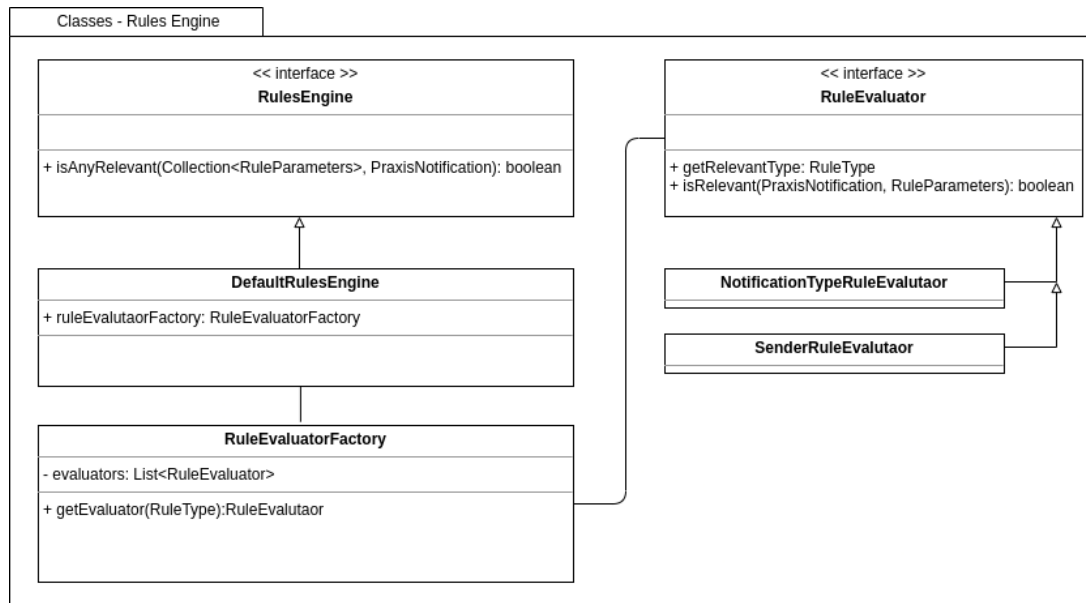


Abbildung 5.11: Klassendiagramm Rules Engine

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

Die Auswertung der einzelnen Regelparameter innerhalb der RulesEngine wird an einzelne RuleEvaluatoren delegiert. Umgesetzt wird dieses Konzept mit einem Strategy Pattern. Wobei das RuleEvaluator Interface das Strategy Interface darstellt. 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 kann. Über die Dependency Injection des Spring Frameworks, kann diese RuleEvaluatorFactory einfach und erweiterbar implementiert werden:


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

Listing 4: RuleEvaluatorFactory.java

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

Domäne Notification

Hier gehts darum Benachrichtigungen zu versenden.



Abbildung 5.12: Domänenmodell Notification

PraxisNotification Jede Notification die der CloudService erhält wird bereits vor dem Versenden persistiert. Dient der Nachvollziehbarkeit. Ermöglicht das Wiederholen von fehlgeschlagenen.

SendNotificationProcess Für jeden Empfänger an den versucht wird eine Notification zu versenden, wird ein SendNotificationProcess erstellt. Dient der Nachvollziehbarkeit. Ermöglicht das Wiederholen von fehlgeschlagenen.

Services Notification

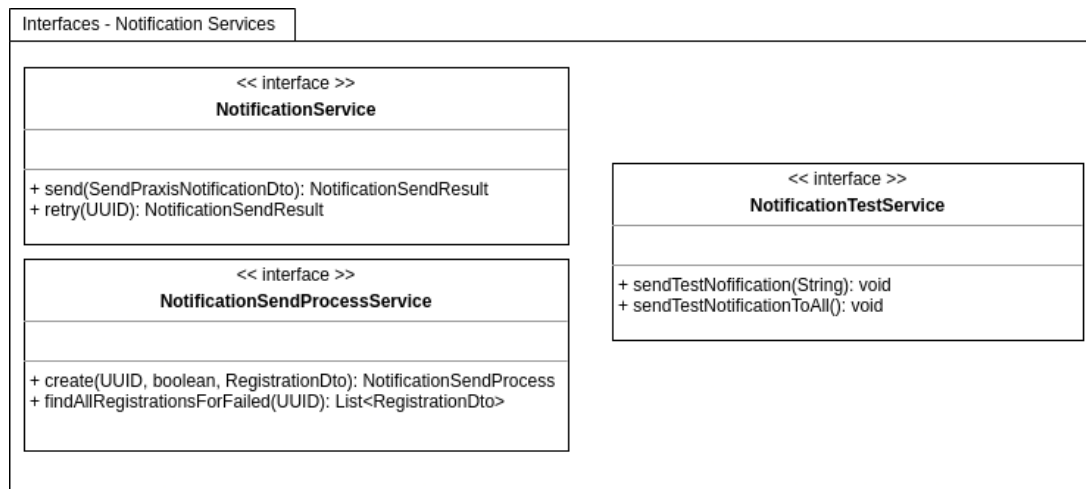


Abbildung 5.13: Klassendiagramm Notification Service Interfaces

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. Im Fall der Wiederholung werden diese Informationen nicht mehr benötigt. Da die Notifikation 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. Die Nachricht die hier versendet wird ist im Cloud Service vorgegeben. Sie beinhaltet lediglich Platzhaltertexte und ist nicht konfigurierbar.

5.3.3 Laufzeitmodell

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

Client Registration

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

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. Sobald eine Konfiguration ausgewählt ist, werden alle dafür Konfigurierten NotificationTypes geladen und im UI die entsprechenden Buttons erstellt. Sobald eine Konfiguration geladen ist, registriert sich der Mobile Client beim Messaging Service. Als Antwort erhält er ein eindeutiges Token, welches verwendet werden kann, um an diesem Client Nachrichten zu senden. Der Mobile Client registriert sich schliesslich mit dem Token vom Messaging Service und der ausgewählten Konfiguration beim Cloud Service. In diesem Zustand ist der Client dem Messaging Service und dem Cloud Service bekannt und ist bereit Benachrichtigungen zu empfangen.

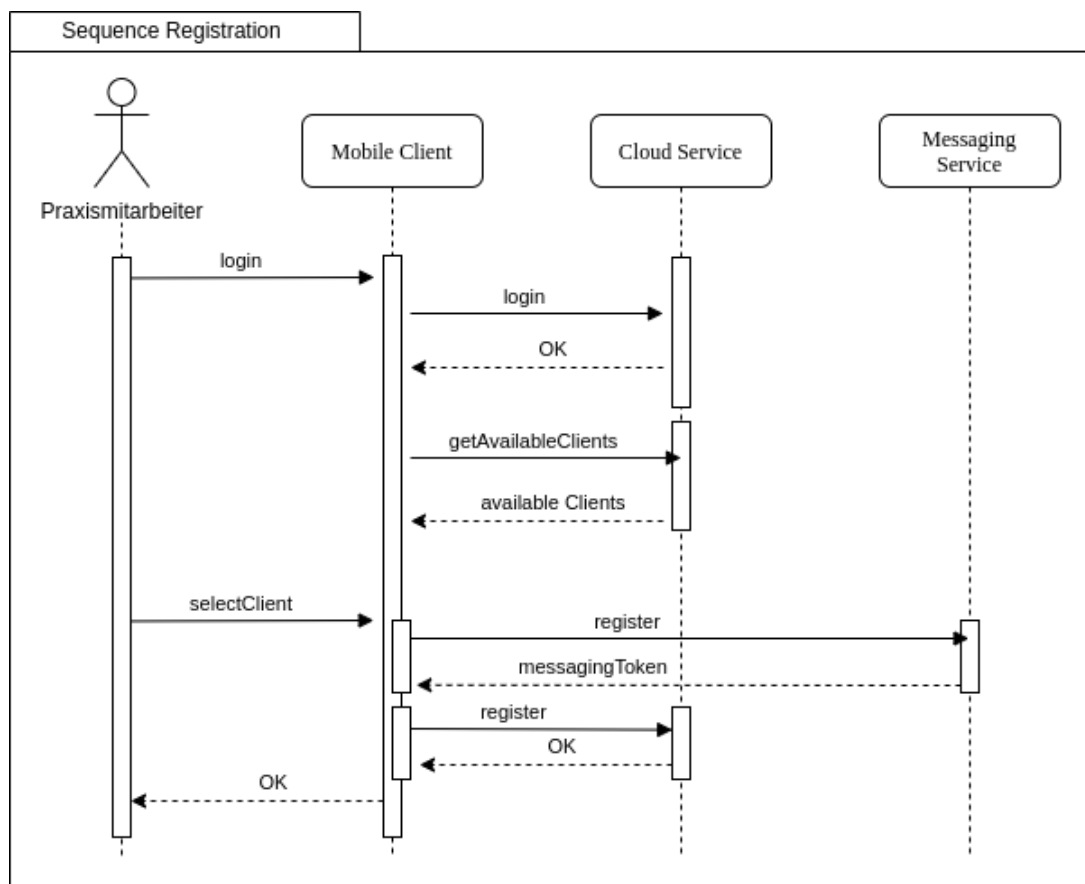


Abbildung 5.14: Ablauf Registration

Benachrichtigung versenden und empfangen

Pre-Condition: Client Registrierung ist für 2 Clients abgeschlossen. Es bestehen gültige Subscriber Confgs. Konfiguration ist geladen.

Der Benutzer tippt auf einen der Benachrichtigungsbuttons. Pro Button ist die Id des verbundenen NotificationTypes hinterlegt. Es wird nun eine Anfrage an den NotificationController im Cloud Service gesendet. Darin enthalten sind die Id des Absender Clients und die Id des NotificationTypes. Der NotificationController macht eine Anfrage an den ConfigurationController um alle relevanten empfänger zu finden. Im ConfigurationController werden alle konfigurierten ClientConfigurations nach Subscriber Regeln evaluiert. Der ConfigurationController gibt schliesslich eine Liste der Registrations zurück die zu einer ClientConfiguration gehören für die eine der Konfigurierten Regeln zugetroffen hat. Der NotificationController lädt den NotificationType aus der Send-Anfrage und benutzt diese Daten um eine Benachrichtigung an alle Empfänger für die er gerade Registrations geladen hat zu senden. Der NotificationController meldet zurück, ob der Versand an alle Empfänger funktioniert hat. Ist dies nicht der Fall wird der Retry-Process auf Client Seite gestartet.

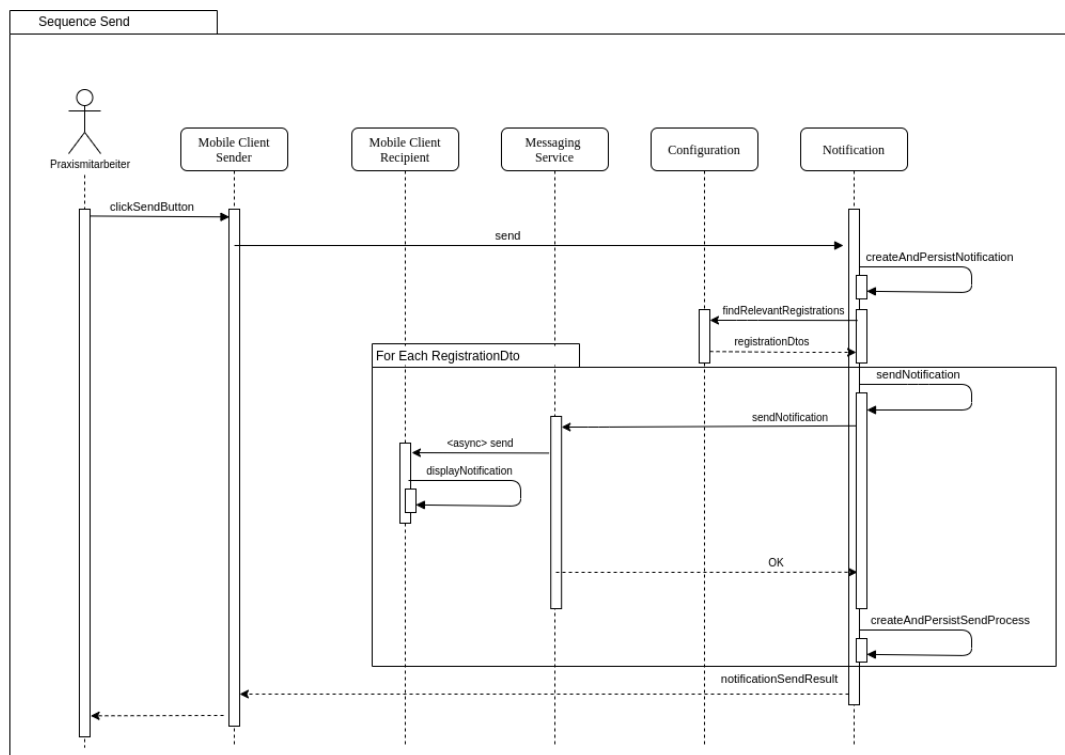


Abbildung 5.15: Ablauf Benachrichtigung Senden und Empfangen

Benachrichtigung wiederholen

Vorbedingung: Es wurde mindestens in Client inklusive ClientConfiguration erfasst und dem Praxismitarbeiter zugewiesen. Es wurde ein zweiter Client inklusive ClientConfiguration erfasst. Der zweite Client wurde so konfiguriert, dass er Benachrichtigungen die vom ersten Client empfängt. Beide Clients laufen und haben sich bei Messaging Service sowie Cloud Service registriert. Der Sender Client hat eine Benachrichtigung versendet. Das Versenden an mindestens einen Client aus dem Cloud Service ist fehlgeschlagen und der Cloud Service gibt ein negatives SendNotificationResult zurück.

Der Client zeigt einen Dialog an in dem der Benutzer informiert wird und gefragt wird, ob er die Fehlgeschlagenen wiederholen möchte. Bestätigt der Benutzer wird eine Retry-Anfrage an den Notification-Controller gesendet. Parameter ist die technische id der Notification die fehlgeschlagen ist. Notification-Controller durchsucht die NotificationSendProcess tabelle nach der gegebenen id und filtert auf fehlgeschlagene. Anschliessend wird der send prozess anhand der tokens in dieser NotificationSendProcess Instanzen wiederholt.

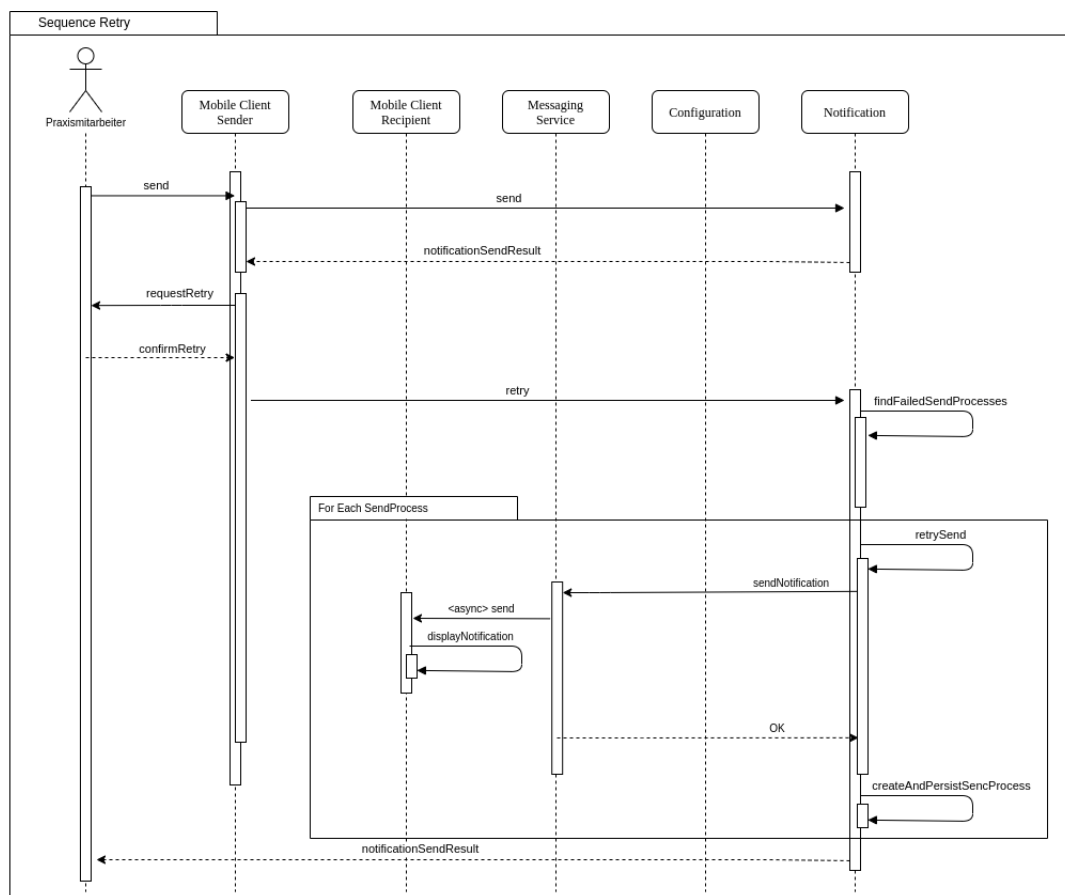


Abbildung 5.16: Ablauf Benachrichtigung Wiederholen

5.3.4 API

Verwaltung

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 Controller 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. Die Registrierung darf nie manuell von einem Administrator erfasst werden. Sie muss immer von einem Mobile Client her kommen der sich beim Cloud Service registriert oder de-registriert. Lesender Zugriff auf die Registrierungen ist nur nötig, wenn der Cloud Service eine Benachrichtigung versendet und die Messaging Tokens der relevanten Empfänger laden muss. Dementsprechend bietet der RegistrationController nicht die volle Verwaltungs API sondern nur folgendes Subset:

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

Authentifizierung

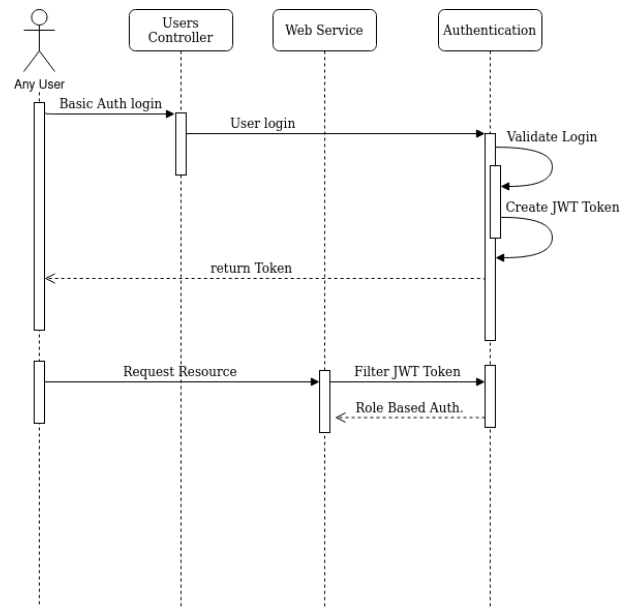


Abbildung 5.17: Authentifizierung-Sequenz

Fachliches

Weiter gibt es Operationen welche eine fachliche Handlung darstellen die nicht der Verwaltung von Konfigurationen dienen. Die API für diese Operationen folgen dem folgenden Schema:

/api/entity-name/action-name

Der Cloud Service bietet die folgenden fachlichen Endpunkte an:

Aktion	HTTP	Pfad	Body	Response
Notifikation versenden	POST	/api/notifications/send	NotificationDto	NotificationSendResult
Notifikation wiederholen	POST	/api/notifications/retry	Notification Id	NotificationSendResult

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 [4].

5.4.2 Anwendung

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

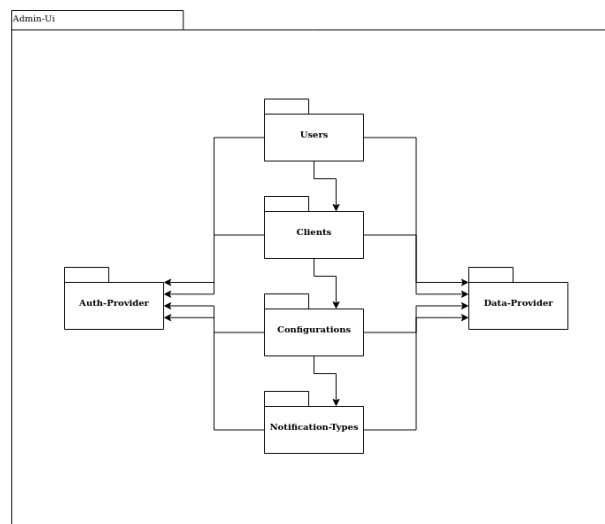


Abbildung 5.18: Admin-Ui Package Diagramm

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 versenden" mit den folgenden Einschränkungen umgesetzt:

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

F02 - Benachrichtigungen empfangen

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

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

F04 - Über Benachrichtigungen Notifizieren

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

- Die Notifizierung erfolgt ohne Audio Signal.

5.5.2 Technische Anforderungen

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

T01 - iPad Client

Der für den Proof Of Concept umgesetzte Client muss auf einem iPad funktionieren und alle Anforderungen die an den Proof Of Concept gestellt werden erfüllen. 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. .

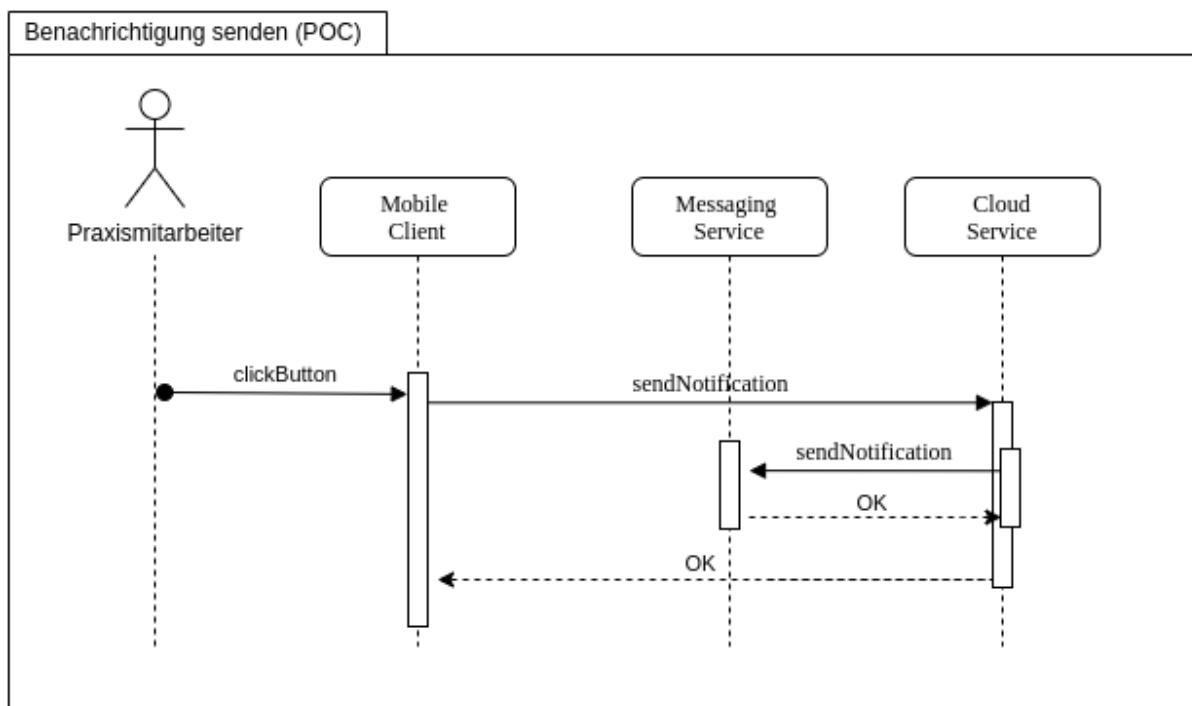


Abbildung 5.19: Proof Of Concept - Benachrichtigung versenden

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.

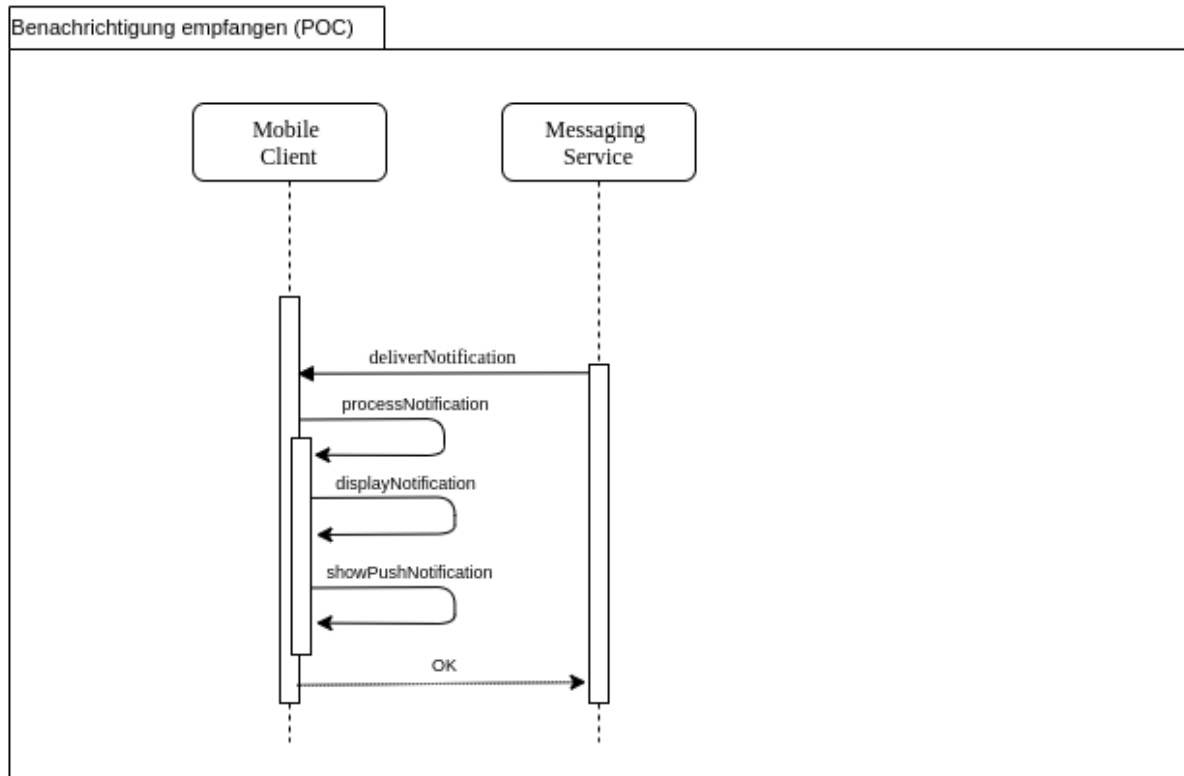


Abbildung 5.20: Proof Of Concept - Benachrichtigung empfangen

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. Über den angebundenen Messaging Service Firebase Messaging ist es möglich, Benachrichtigungen zwischen Mobile Clients zu versenden. Cloud Service und Admin UI ermöglichen es dabei die Benachrichtigungen die Versendet werden können und welcher Client welche Benachrichtigungen erhalten soll zu konfigurieren. 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 kann¹.

Im Rahmen des Projektes wurden damit die Milestones M01 bis M06² erreicht. Umgesetzt wurden die Milestones mit den folgenden User Stories inklusive aller dazu definierten Features und Szenarien³:

- 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 Milestones M06 bis M10 konnten im Rahmen dieses Projektes nicht umgesetzt werden. Damit sind folgende User Stories ausserhalb des Projektrahmens gefallen:

- U08 - Physischer 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

6.1.1 Mobile Client

Dieses Kapitel zeigt die umgesetzten Ansichten des Mobile Clients. Eine detaillierte Beschreibung, wie der Mobile Client bedient werden kann, befindet sich im Anhang der Projektdokumentation.

¹Siehe Anhang Betriebshandbuch

²Siehe Kapitel 2.2

³Siehe Kapitel 3

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.

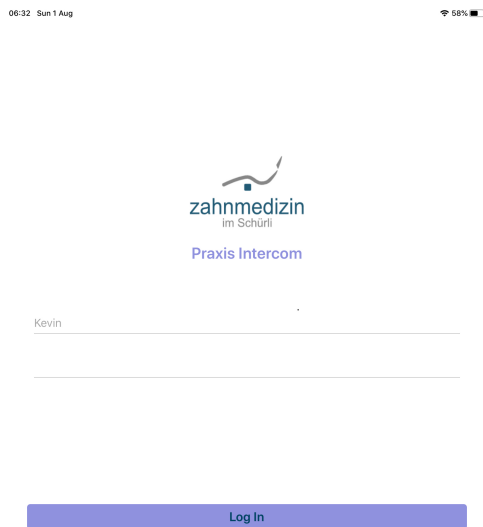


Abbildung 6.1: Login

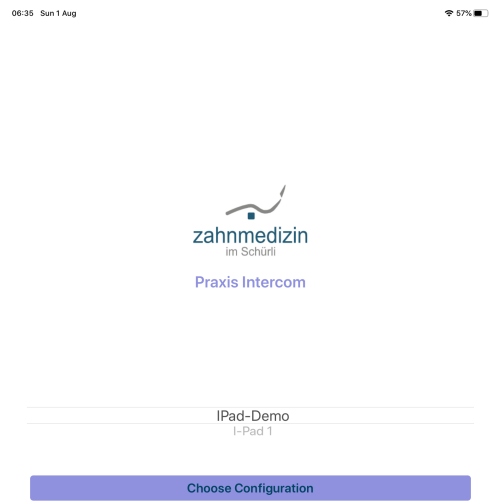


Abbildung 6.2: Konfiguration

Benachrichtigungen versenden

Im Tab Home der Startseite werden Buttons angezeigt um Benachrichtigungen zu versenden. Die Buttons werden hier dynamisch aus der geladenen Konfiguration angezeigt. Diese Konfiguration beinhaltet den Text der auf dem Button angezeigt wird sowie den Inhalt der Benachrichtigung versendet wird.

Klickt der Benutzer auf einen der Buttons wird die entsprechende Benachrichtigung versendet. Die Vermittlung, wem diese Benachrichtigung zugestellt wird, liegt bei dem Cloud Service. Dieser entscheidet Anhand der registrierten Clients und den vorhanden Konfigurationen, welche Clients Empfänger für diese Benachrichtigung sind. Schlägt das Versenden der Benachrichtigung an mindestens einen Empfänger fehl, wird dies dem Benutzer angezeigt. Er hat dann die Möglichkeit, das Versenden an diese Empfänger wiederholen.

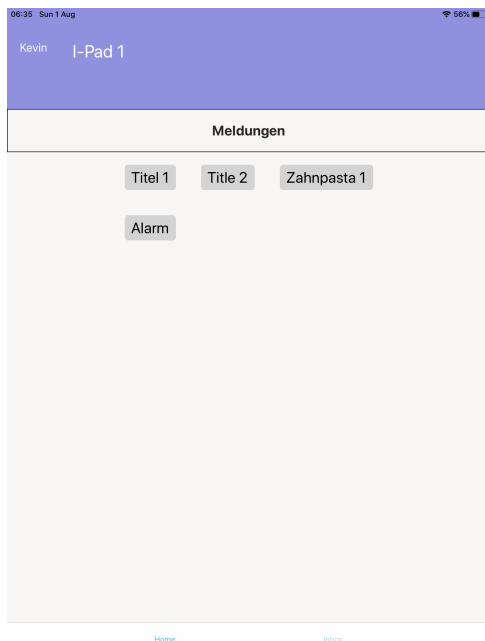


Abbildung 6.3: Home

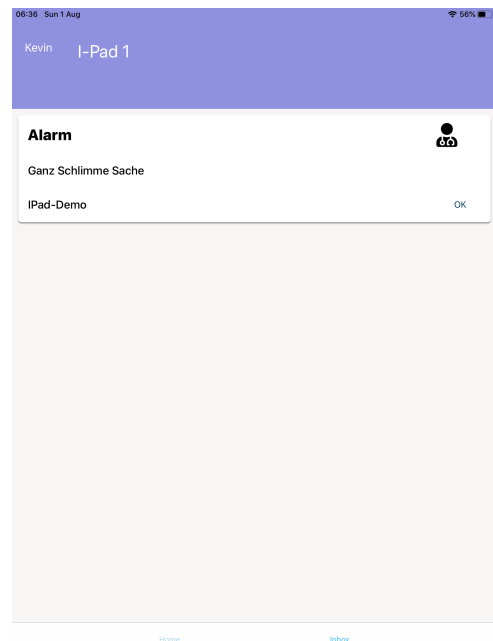


Abbildung 6.4: Retry

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. Siehe Abbildung xx. Auch wenn die Benachrichtigung im Hintergrund empfangen wurde, wird diese in der Inbox angezeigt.

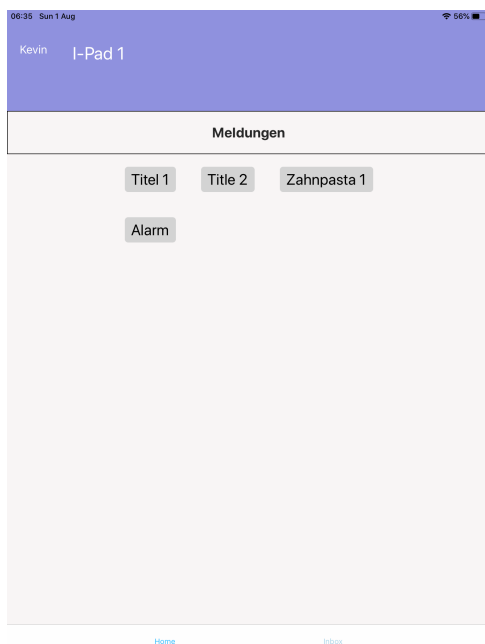


Abbildung 6.5: Inbox

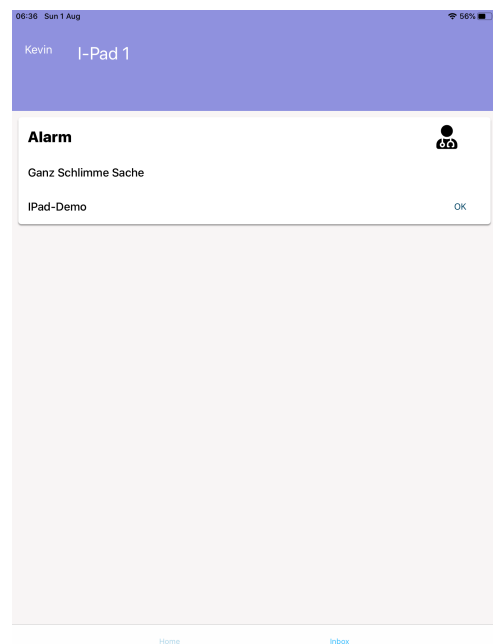


Abbildung 6.6: Push Benachrichtigung

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 und 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 unter www.praxisruf.ch/api an. Diese API ist unter www.praxisruf.ch/api erreichbar und über Basic Authentication geschützt.

Zu Test- und Entwicklungszwecken ist die API zudem über www.praxisruf.ch/swagger-ui.html erreichbar. Das Swagger Interface dokumentiert die API die der Cloud Service anbietet. Dies beinhaltet die verfügbaren Endpunkte und das Model welches für Input und Output Werte dieser API dienen.

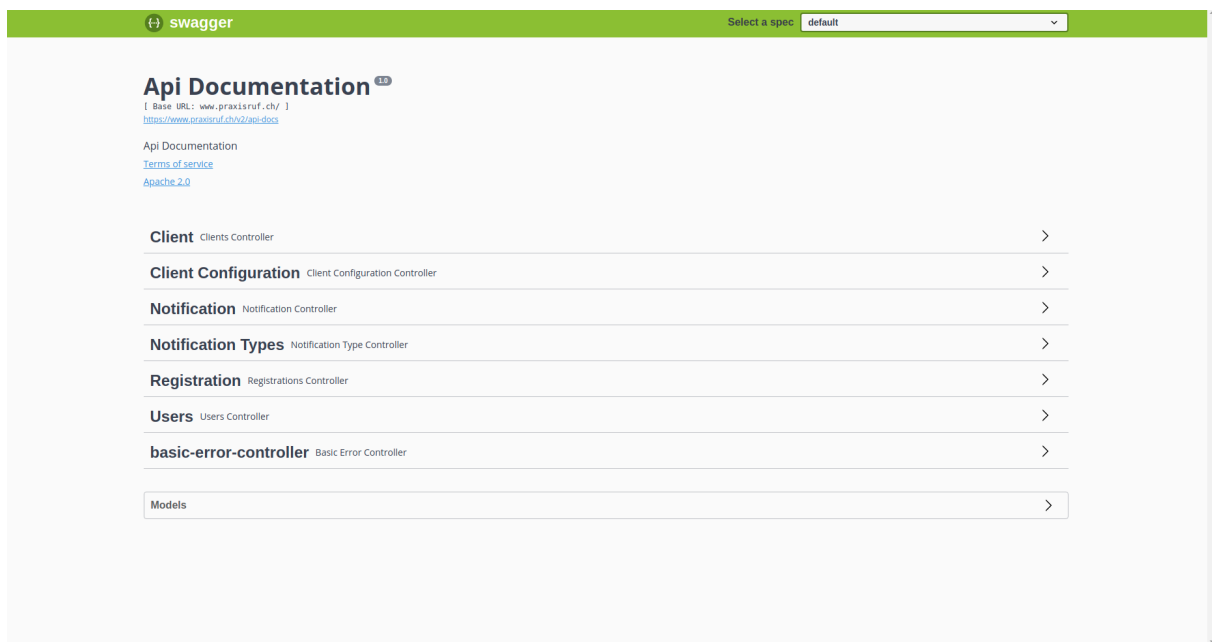


Abbildung 6.7: Swagger UI

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 Praxisverantwortliche das Admin UI verwenden soll, ist die Benutzeroberfläche durch ein Login geschützt. Die entsprechenden Anmeldeinformationen müssen bei Installation des Cloud Services vom Betreiber manuell konfiguriert werden. Das Admin UI beinhaltet vier Bereiche.

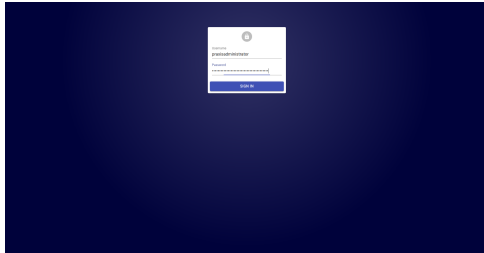


Abbildung 6.8: Login

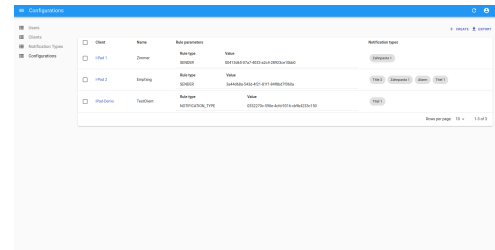


Abbildung 6.9: Configuration Overview

Der Bereich **Users** dient dazu Benutzer zu erstellen, welche sich als Benutzer am Mobile Client anmelden können.

Unter **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 **Configurations** kommen schliesslich alle Teile zusammen. Hier kann die Konfiguration für einen Client erfasst werden. Dabei wird die Konfiguration einem Client zugewiesen. Die Konfiguration hat eine Liste von Notification Types, welche auf dem zugewiesenen Mobile Client als Versenden Button angezeigt werden. Weiter beinhaltet die Konfiguration eine Liste von Regel Parametern, welche bestimmen, welche Benachrichtigungen dem zugewiesenen Client weitergeleitet werden.

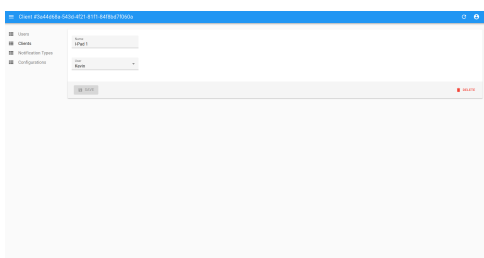


Abbildung 6.10: Login

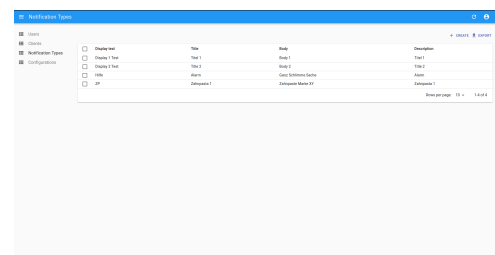


Abbildung 6.11: Configuration Overview

Jeder der Bereiche im Admin UI bietet dem Benutzer die Möglichkeit, dieses Konfigurationsobjekt anzuzeigen, erstellen, bearbeiten und löschen. Auf der Startseite jedes Bereiches wird eine Liste mit allen relevanten Einträgen angezeigt. Über die Schaltfläche 'Create' kann der Benutzer neue Elemente erfassen. Durch einen Klick auf ein einzelnes Element, kommt der Benutzer auf eine Ansicht, wo das Element bearbeitet werden kann.

6.2 Tests

Benutzertests

Testablauf

Am xx.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.

Als 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 Benachrichtigungen 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	+

Performancetests

1. Nur ein pyhsisches Gerät 2. Keine Zeit gehabt.

6.3 Herausforderungen

Cloud Service

Eine grosse Herausforderung war das Konzept für den Cloud Service. Dieser musste ein konfigurierbares Subscription System bieten über den die Mobile Clients Benachrichtigungen versenden 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 geringen Aufwand weitere Regeln zu implementieren.

IOS Entwicklung

Eine weitere Herausforderung im Projekt war die Entwicklung und Konzeptierung des Mobile Clients. Wir haben beide keine vrogängige Erfahrung der Entwicklung von Mobilen Applikationen für Andorid oder IOs. Dementsprechend mussten wir teilweise mehr Aufwand in Recherche und Entwicklung stecken als erwartet. Hinzu kam, dass die Recherche für die zu Verwendenden Technologien im Mobile CLient sich als aufwändiger als erwartet dargestellt hat. Dies insbesondere wegen der Bedingung, dass die gleiche Code Basis für Andorid und IOS verwendet werden kann und den Hindernissen die IOS für die meisten solchen Frameworks bietet. Insbesondere haben es Einschränkungen für Background Aktivitäten **TODO: CITATION** und andere Barrieren im IOS Betriebssystem schwer gemacht, Messaging Services anzubinden und Push Benachrichtigungen zu versenden. Wegen diesen Barrieren waren viele der Ansätze die wir als passend und effizient gewertet haben schlicht keine Option.

In der Umsetzung war das wenig anders. Mit der gewählten Technologie Native Script konnten wir eine getilete Code Basis haben. Allerdings musste die Entwicklungsumgebung schlussendlich genauso aufgesetzt werden wie es für die native Entwicklung nötig gewesen wäre. Vom Effizienzgewinn den Nativescript in längeren Projekten bieten könnte, konnten wir deshalb nicht profitieren. Die Interaktion zwischen Nativeskript und dem IOS funktioniert über plugins. **TODO: CITATION** Es gibt für fast alle Funktionen plugins. Diese sind aber oft unvollständig implementiert oder funktionieren nur mit spezifischen Versionen von NativeSkript / IOS.

AWS Infrastruktur

Ein weiterer grosser Zeitfresser war das Aufsetzen der AWS Infrastruktur. Es war uns so schlussendlich Möglich die ganze Infrastruktur die benötigt wurde aufzusetzen. Da wir beide wenig bis keine Erfahrung mit dem Aufsetzen von Cloud INfrastrukturen und CI/CD Pipelines haben, hat das allerdings eine Weile gedauert. AWS bietet für alles Funktionen die wir benötigt haben ausführliche Dokumentation. Das hat die UMsetzung deutlich vereinfacht. Die Hürde das Ganze zuv erstehenund Umzusetzen musste trotzdem zuerst einmal genommen werden. Insbesondere das Aufsetzen von Elastik Beanstalk mit einem Vorgeschalteten Load Balancer und HTTPS Konfiguration sowie die Anbindung an eine Relationale Datenbank hat viel Zeit gekostet.

API Konzept

Zum Anfang der Konzept Phase wurde zu wenig Zeit für das planen der API nach aussen umgesetzt. Es wurde zuerst alles als ein einzelner Endpunkt dargestellt. Intern gab es nur zwei Services für die Logik. Die auszuführende Logik war insbesondere dank des Admin UIs umfassender als zuerst geplant. Dementsprechend musste die API aufgeteilt werden, um die Weiterentwicklung und Wartbarkeit des Projektes zu gewähren. Diese Umstellung hat allerdings wieder Zeit gebraucht.

Projektplanung

Geplant war, dass die Milestones aus der Projektplanung als Roten Faden durch das Projekt führen. Der agile Ansatz hat geholfen, dass wir immer wussten, was als nächstes zu tun ist. Eine leicht feingranularere Planung von Anfang an, hätte aber geholfen besser einzuordnen, wie wir im Projektplan stehen. So hätte man früher sehen können für was es nicht reicht entsprechende Massnahmen ergreifen können.

6.4 Lessons Learned

Vollständige Konzepte

Die Konzepte für Mobile Client und Cloud Service sind teilweise während der Entwicklung entstanden. So hat sich z.B. die API nach aussen während des Projektes durch Refactorings geändert, weil Anpassungen nötig werden. Im Konzept fehlten am Anfang auch ein konkreter Plan für den Betrieb mit AWS. Wir haben gelernt, dass es sich bezahlt macht alle Konzepte von Anfang an zu haben. Diese Konzepte müssen dabei nicht von Anfang an ausgereift sein. Sie dienen als Startpunkt und Orientierungshilfe und können sich während des Projektes anpassen. Für dieses Projekt wäre es insbesondere gut gewesen, die nötigen AWS Services bereits in ein Konzept aufzunehmen.

Projektplanung

Wir haben gute Erfahrungen damit gemacht, die Prioritäten vorzu mit dem Kunden abzusprechen und umzusetzen. Rückblickend wäre es aber für diese Planung und Sitzungen hilfreich gewesen, einen konkreteren Plan zu haben. Die Meilensteine die zum Anfang definiert wurden haben einen groben Überblick gegeben. Es wäre aber hilfreich für die Umsetzung gewesen, von Anfang an schon konkretere Anforderungen zu haben. Für dieses Projekt hätte es sich z.B. angeboten nach der Evaluierung der Technologien die Anforderungen bereits detaillierter festzuhalten. So hätte sich die Umsetzung effizienter gestalten lassen und man hätte mehr Features umsetzen können.

Ein weiterer Punkt bei der Projektplanung ist, dass das Schreiben des Projektberichts überhaupt nicht im Projektplan eingeplant war. Dementsprechend war das Berichtschreiben spät dran und aufwändig. Für künftige Projekte würden wir auf jeden Fall Zeit für den Bericht bereits im Plan einplanen. Und dies über die ganze Dauer des Projekts und nicht erst am Ende.

Go Native

Eine geteilte Codebasis für mehrere Plattformen kann Zeit bei der Entwicklung und Wartung sparen. Es bringt aber den Nachteil, dass die Applikation komplizierter wird, weil zwischen den unterstützten Betriebssystemen unterschieden werden muss. Weiter erschwert es die Verwendung von Betriebssystemfunktionen und Gerätehardware, da diese für alle Betriebssysteme abstrahiert werden müssen. Schlussendlich bringt es eine zusätzliche Schicht in die Architektur, bei der Kompatibilität eine grosse Rolle spielen kann. Für Applikationen die keine oder nur wenig mit direkt mit dem Betriebssystem interagieren kann eine geteilte Code Basis unter dem Strich ein Vorteil sein. Sobald aber kritische Teile der Applikation mit Betriebssystemnahen Funktionen zusammenhängen, empfehlen wir mehrere native Applikationen zu entwickeln. So entsteht teilweise doppelter Code. Dieser kann aber einfacher erstellt und gewartet werden und bietet bessere Kompatibilitätsgarantie.

7 Schluss

Bis zum Ende des Projekts konnte das Praxisrufsystem umgesetzt werden. Für das Praxisrufsystem wurden eine Mobile Applikation, ein Backend Service und eine Web-Applikation konzipiert und umgesetzt. Mit Firebase Messaging wurde ein externer Messaging Provider angebunden über den Serverseitige Benachrichtigungen versendet werden können. Weiter wurde Betriebs- und Entwicklungsinfrastruktur mit Amazon Webservices aufgebaut. Das umgesetzte System bietet die Möglichkeit Benachrichtigungen zwischen Mobile Clients zu versenden.

Insgesamt sind wir zufrieden mit den Konzepten und Ergebnissen die wir geliefert haben. Wir sind insbesondere mit dem Konzept für die Registrierung von Clients bei Messaging und Cloud Service und dem Konzept für die RulesEngine die es Clients ermöglicht Benachrichtigungen zu abonnieren. Leider konnte das Praxisrufsystem wegen diverser Herausforderungen⁴ nicht im Umfang wie es in der Aufgabenstellung beschrieben wurde umgesetzt werden. Dementsprechend sind wir in den Punkten Reifegrad und Funktionsumfang nicht gänzlich zufrieden mit den Resultaten.

Wir sind der Ansicht, das bei besserer Projektplanung einige zusätzliche Anforderungen umgesetzt hätten werden können. Allerdings bleibt zu beachten, dass wir beide keine Erfahrung mit Amazon Web Services oder der Entwicklung von Mobilen Applikationen haben. Wir sind fälschlicherweise davon ausgegangen, dass es Möglich sein wird die Anforderungen des Projektes mit uns vertrauteren Web Technologien umzusetzen. Dieses Defizit wäre auch bei besserer Planung da gewesen.

Schlussendlich sind wir froh, um die Erfahrungen die wir aus diesem Projekt mitnehmen können und nehmen die Lessons Learned⁵ daraus in nächste Projekte mit. Wir sind uns bewusst, dass das System in dieser Form nicht alle Bedürfnisse des Kunden abdecken kann. und hätten uns gewünscht, mehr der fehlenden Funktionen umsetzen zu können und mehr Zeit in die Optimierung der Applikationen stecken zu können. Angesichts der Herausforderungen sind wir mit dem Ergebniss als Ganzes aber zufrieden und zuversichtlich, dass wir ein gutes System gebaut haben.

⁴Siehe Kapitel x.y

⁵Siehe Kapitel x.y

Literaturverzeichnis

- [1] O. Foundation. (). How NativeScript Works, Adresse: <https://v7.docs.nativescript.org/core-concepts/technical-overview>.
- [2] —, (). What is iOS Runtime for NativeScript? Adresse: <https://v7.docs.nativescript.org/core-concepts/ios-runtime/overview>.
- [3] S. Odean. (). Software Architecture — The Onion Architecture, Adresse: <https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec>.
- [4] Marmelab. (). react-admin, Adresse: <https://marmelab.com/react-admin/Readme.html>.


Abbildungsverzeichnis

0.1	Titlebild	1
2.1	Projektplan	4
5.1	System	15
5.2	NativeScript-Overview	17
5.3	Mobile-Client Package Diagramm	20
5.4	Mobile-Client Flow Chart	21
5.5	HomeScreen Mockup	22
5.6	Inbox Mockup	22
5.7	Clean Architecture	24
5.8	Package Struktur Cloud Service	25
5.9	Domänenmodell Configuration	26
5.10	Klassendiagramm Configuration Service Interfaces	27
5.11	Klassendiagramm Rules Engine	28
5.12	Domänenmodell Notification	30
5.13	Klassendiagramm Notification Service Interfaces	31
5.14	Ablauf Registration	32
5.15	Ablauf Benachrichtigung Senden und Empfangen	33
5.16	Ablauf Benachrichtigung Wiederholen	34
5.17	Authentifizierung-Sequenz	36
5.18	Admin-Ui Package Diagramm	38
5.19	Proof Of Concept - Benachrichtigung versenden	40
5.20	Proof Of Concept - Benachrichtigung empfangen	41
6.1	Login	43
6.2	Konfiguration	43
6.3	Home	44
6.4	Retry	44
6.5	Inbox	45
6.6	Push Benachrichtigung	45
6.7	Swagger UI	46
6.8	Login	47
6.9	Configuration Overview	47
6.10	Login	47

6.11 Configuration Overview	47
8.1 Aufgabenstellung	56

8 Anhang

8.1 Aufgabenstellung



Fachhochschule
Nordwestschweiz

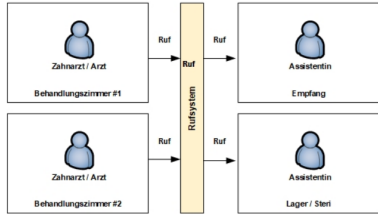
App
Web

21FS_IMVS01: Cloudbasiertes Praxisrufsystem

Betreuer: Daniel Jossen	Priorität 1 Arbeitsumfang: P5 (180h pro Student)	Priorität 2 ---
Sprachen: Deutsch	Teamgrösse: 2er Team	---

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

Technologien/Fachliche Schwerpunkte/Referenzen

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

8.2 Benutzerhandbuch

8.3 Betriebshandbuch

8.4 Entwicklerdokumentation

8.5 Ehrlichkeitserklärung