Mobile Anwendungen

Dokumentation
Gruppe M

Raphael Barth Mike Sickmüller
Marius Cerwenetz

Mannheim, Sommersemester 2020

Inhaltsverzeichnis

1 Beschreibung 1
2 Architektur 1
3 PDF 1
4 RoomLifecycleService 1
5 Tag - QR und NFC 3
6 MQTT Service 3
7 Datenbank 6
8 Fazit 6
9 Ausblick 6

1 Beschreibung

*[1]

2 Architektur

3 PDF

Das System soll dem Nutzer die Möglichkeit bieten Informationen über eine raum als PDF zu teilen. Dabei gibt es zwei unterschiedliche PDF‘S, diese beinhalten die generellen raum Informationen sowie entweder den dazu gehörigen QR-Code oder eine Liste aller Teilnehmer. Das Android Framework stellt zum Erstellen von PDF‘S die PDFDokument Klasse bereit in dieser wir eine Seite mit den entsprechenden Formatierungen erstellt. Für eine A4 Seite nach DIN Norm ergeben sich die werte 846x594pixel näheres hierzu zu Berechnung ist unter developer.android zu finden. Die erstellte Seite kann mittels eines Canvas editiert werden. Formatierung einstellengen der Texte und Grafikelemente lassen sich über eine Paint Klasse einstellen. Das erstellt PDFDokument wird im externen Datei Verzeichnis der App abgespeichert. Der Nutzer besitzt bei einem fehlgeschlagenen teilen somit trotzdem die Möglichkeit auf das erstellte PDF zuzugreifen. Um das PDF zu teilen wird ein FileProvider verwendet welcher im Manifest deklariert ist. Der FileProvider bietet dem Nutzer einer Auswahl der installierten Anwendungen über welcher er das PDF teilen kann.

Für eine spätere App Version ist das PDF mit der Übersicht aller Teilnehmer zu überarbeiten. In der aktuellen Version wird die E-Mail-Adresse ab einer bestimmten länge zum teil mit in die nächste spalte geschrieben. Ziel ist es das ab einer bestimmten länge ein automatischer Zeilenumbruch in der entsprechenden Spalte stattfindet.

4 RoomLifecycleService

Der RoomLifecycleService steuert, wie der Name bereits andeutet, in welchem Zustand sich Räume befinden. Er wird ausschließlich im Host-Modus betrieben *[2] . Die Entscheidung fiel auf einen Service, weil wir auch im nicht-geöffneten Zustand der App auf Nachrichten von außen reagieren wollte. Beispiel: Der Host eröffnet einen Raum und wechselt die App oder sperrt

*[1] einheitliche schreib weise für klassen namen evtl. kursive

*[2] passt nicht ganz wird au im participant modus genutzt

das Smartphone. Wenn die Kontaktverfolgungsapp nun nicht mehr im Vordergrund ist während das Timeout ausläuft oder der Raum in die Öffnungszeit reinrutscht, dann würde er nicht geschlossen respektive geöffnet werden. Um also zu vermeiden dass der Host über die gesamt Zeit sein Smartphone nicht verwenden kann und das Display mit hohem Stromverbrauch den Akku entleert haben wir uns für einen Service entschieden.

Wird der Service gestartet wird die Raumüberprüfungsroutine nebenläufig in einem neuen Thread gestartet. Dadurch verhindern wir dass das der Hauptthread machen muss. Der Raumstatus wird jede Sekunde mit einem Busy-Wait-Algorithmus überprüft. Das sollte man nicht in den Hauptthread auslagern, da er diesen unnötig blockiert. Der Datenbankzugriff ist nicht die Ursache für die Designentscheidung, denn dadurch dass wir schon über das Repository nebenläufig auf die Datenbank zugreifen sind wir bezüglich Datenbank-Latenz bereits auf der sicheren Seite. Im Thread überprüfen wir initial ob aktuell geschlossene Räume geöffnet und aktuell geöffnete Räume geschlossen werden sollten. Nebenher wird im Mainthread versucht den MQTT-Service zu einzubinden. Denn der Host soll natürlich auf alle Räume die sich öffnen hören bzw. aufhören auf alle Räume zu hören die sich geschlossen haben. Außerdem muss der Host den Teilnehmern die Raumeigenschaften und den Öffnungsstatus beim Öffnen mitteilen. Wir brauchen den MQTT-Service hier also um Raumstatusänderungen den Teilnehmern mitzuteilen. Zurück zum LifeCycle-Service. Ein Raum hat drei mögliche Zustände:

1. Wird sich öffnen
2. Geöffnet
3. Geschlossen

Jede Sekunde wird also überprüft ob Räume die geschlossen sind geöffnet werden sollen, ob Räume die sich öffnen werden geöffnet werden können und ob Räume die geöffnet sind geschlossen werden sollten. Falls noch Teilnehmer in zu schließenden Räumen sind werden Sie rausgeworfen. In der Datenbank wird dann für alle Teilnehmer als Austrittszeitpunkt die Timeout-Zeit des Raums eingetragen. Auf der Teilnehmerseite reagiert der Observer auf die Änderung des Raumstatuses und beendet den Raum auch real für den Teilnehmer.

5 Tag - QR und NFC

6 MQTT Service

MQTT Service Das MQTT Protokol dient als Der MQTT Service wird in der StartActivity gestartet und beendet alle anderen Activitys binden nur auf den MQTTService. Das Binden des MQTTServices findet in den Acticvities 11_EnterViaNfcQr, 14_RoomPArticipntsDetails, 22_RoomHostDetails statt. Des Weiteren wird der Service in dem RoomLiveCycle Service gebunden näheres hierzu unter Abschnitt. Beim verwenden des MQTT Services wird in jeder Activity geprüft ob dieser schon gebunden ist (not null) ist dies nicht der Fall wird das zu sendende Object zwischen gespeichert und automatisch in Anschluss an das binden gesendet. Die mittels MQTT Service zu nutzenden Methoden sin ausführlich im Code mittels JavaDoc beschrieben.

Allgemeines Nachrichten Format Die Kommunikation zwischen dem Gastgeber und den Teilnehmern verläuft beim MQTT Protokoll über sognannte Topics hierzu siehe Abbildung. Für einen Raum werde immer 2 Topics benötigt. Der Teilnehmer kann einem Raum über die Topic moagm/Room-Tag beitreten. Der Gastgeber sendet die nötigen Informationen für die Teilnehmer über die Topic moagm/Room-Tag/public an alle Teilnehmer. Durch diese Trennung der Kanäle wird verhindert das Teilnehmer nach richten von anderen Teilnehmern an den host empfangen.

Insgesamt gibt es vier verschiedene Nachrichtentypen welche zwischen den Teilnehmern und dem Gastgeber ausgetauscht werden. Alle Nachrichten werden im JSON- Format versendet. Dabei kann man die Nachrichten in zwei gruppen unterteilen. Die Nachrichten anmeldung.JSON, abmeldung.JSON werden vom Teilnehmer an den Host gesendet. Teilnermer.JSON und Rauminfo.JSON sendet der Host an alle Teilnehmer. Der Aufbau der Aufbau der einzelnen JSON Formaten ist in Abbildung 1 zu sehen. Mittels eines Konverters werden die entsprechenden JSON Objekte erzeugt und wieder in verwendbare Datentypen zurück gewandelt. Zum identifizieren eines Nachrichten Types wird (warum auch immer..) nicht das Type Feld benutzt. Das Identifizieren wird über die Schlüsselworte ENTERTIME (anmeldung.JSON), EXITTIME (abmeldung.JSON), RAUM (raumInfo.JSON) sowie TEILNEHMERLIST (teilnehmer.JSON) durchgeführt. Abhängig von dem empfangen Datentyp werden unterschiedliche Aktionen durch geführt, welche im Folgenden näher erläutert werden. Mit der Listing 1 dargestellten anmeldung.json meldet sich an Teilnehmer beim Gastgeber an und stellt eine Verbindung zu der entsprechende

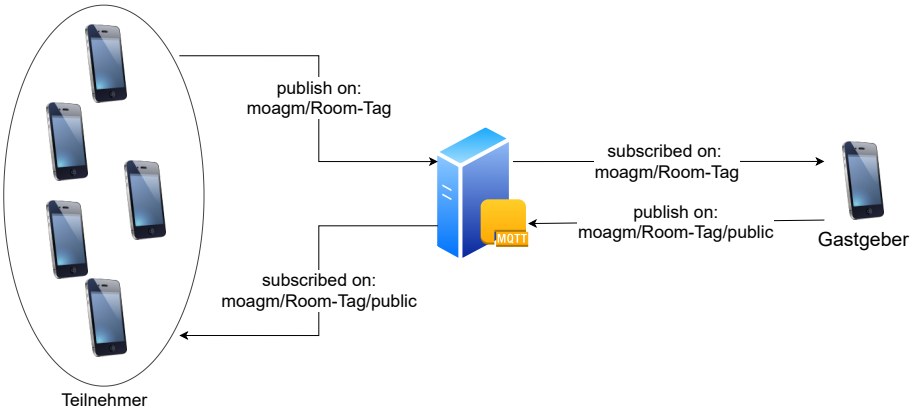

Abbildung 1: Darstellung des MQTT Topics.

Raum Topic her. In der Nachricht sind Alle relevanten Informationen des Teilnehmers enthalten sowie die Uhrzeit wann er den Raum betreten hat. Der MQTT Service konvertiert das erhaltene JSON mittels der AdapterJsonMqtt Klasse in den passenden daten Typ. Das erhalte Teilnehmerobject wird Inder Datenbanktabelle dbParticipant mit der ID des entsprechenden Raumes abgespeichert. Die entsprechende RaumID kann das Topic entnommen werden, auf über welche das JSON object empfangen wurde. Im Anschluss wird an alle Teilnehmer ein Teilnehmer.json sowie Rauminfo.json gesendet.

$*^3$

Die Nachricht abmeldung.json dargestellt in Listing 2 sendet der Teilnehmer an den Gastgeber sobald er den Raum verlässt. Dabei meldet sich der Teilnehmer ebenfalls von der entsprechen raum topic der MQTT Verbindung ab. Analog zum anmelden.json enthält diese Nachricht alle wichtigen Informationen über den Teilnehmer sowie die Uhrzeit zu welcher er den Raum verlässt. Im MQTT Service wird mithilfe des erzeugten Teilnehmer Objektes und der RaumID die Exittime des dazugehörenden Teilnehmers aktualisiert.

$*^3$ passenden titel für Abbildung 1 finden

Listing 3 stellt die Nachricht vom Tye Rauminfo.JSON da. Diese Nachricht enthalten sämtliche Informationen über einen Raum. Jede Vom Teilnehmer Empfangen Rauminfo.json wird in ein RaumObjcet konvertiert und in die Datenbank Tabelle dbRoom eingefügt bzw. aktualisiert den bereits vorhanden Eintrag. Wenn der Gastgeber eine RAuminfo.json versendet welche den Teilnehmern mitteilt das derRuam nun geschlossen ist meldet der MQTT ervice den Gastgeber automatisch von der entsprechden MQTT Topic ab.

Der in Listing 4 dargestellte Nachrichtentype Teilnehmer.JSON beinhaltet eine Liste aller Teilnehmer eines Raumes. Bei Empfangen der Nachricht wird das darin enthaltene JSONArray in eine Liste von Participants Konvertiert. Die

Participants der Liste werden mit der Entsprechden RaumID welche über die Topic ermittelbar ist in die Datenbanktabelle dbParticipants eingefügt. Um zu verhindern das Teilnehmer mehrfach eingetragen werden wird die anzahlt der Bereits früher eingefügten Teilnehmer des raumes ermittelt. Aus der Liste der erhaltenen Teilnehmer werden nur participants eingefügt welche einen mindestens gleich großen Index besitzen wie die ermittelte anzahlt. Diese Variante funktioniert da sich die Reihenfolge der Teilnehmer sich in der Datenbank nie ändert.

Listing 1: anmeldung.json

```
{
 "TYPE": "LOGOUT",
 "TEILNEHMER": {
        "NAME": "",
        "EXTRA": "",
        "EMAIL": "",
        "PHONE": ""},
 "EXITTIME": 0
}
```

Listing 2: abmeldung.json

```
{
 "TYPE": "LOGIN",
 "TEILNEHMER": {
        "NAME": "",
        "EXTRA": "",
        "EMAIL": "",
        "PHONE": ""},
 "ENTERTIME": "0"
}
```

Listing 3: rauminfo.json

```
{
 "TYPE": "RAUMINFO",
 "RAUM": {
        "ID": 0,
        "ROOMNAME": "",
        "STATUS": "",
        "HOST": "",
        "EMAIL": "",
        "PHONE": "",
        "PLACE": "",
        "ADDRESS": "",
        "EXTRA": "",
        "ROOMSTARTTIME": 0,
        "ROOMENDTIME": 0}
}
```

Listing 4: teilnehmer.JSON

```
{
 "TYPE": "TEILNEHMER",
 "TEILNEHMERLIST": [
        {
         "NAME": "",
         "EXTRA": "",
         "EMAIL": "",
         "PHONE": "",
         "ENTERTIME": 0,
         "EXITTIME": 0},
        ...
 ]
}
```

7 Datenbank

8 Fazit

9 Ausblick

Sonstiges Die App ist in der Lage bei einem Verlust der Internetverbindung dem Nutzer ein Feedback mit zu teilen und ein hartes Abbrechen zu verhindern. Dafür wird ein BroadcastReciever verwendet welcher Auf Netzwerkänderungen hört. Der BroadcastReciver wird beim Erstellen des RommLivecyclesServices an diesen gebunden. Bricht die Internetverbindung weg werden alle Services gestoppt. Des Weiteren wird dem Nutzer mitgeteilt das die Internetverbindung verloren wurde und er muss die App mit einer bestehen Internetverbindung neu starten muss.

Sämtliche benötigten Informationen über den Nutzer der App werden mittels Preferences erfasst. Ohne das vollständige Ausfüllen der Preferences ist in Verwenden der App nicht möglich. Um später einen eleganten Zugriff auf die Preferences zu erhalten ist eine MySelf Klasse implementiert.