Mobile Anwendungen

Dokumentation
Kontaktverwaltung App

Gruppe M

Raphael Barth Mike Sickmüller Marius Cerwenetz

Mannheim, Wintersemester 20/21

Zusammenfassung

Die folgende Dokumentation gibt einen Einblick über den Aufbau der Kontaktverwaltung-App, ihrer Komponenten, Designentscheidungen und Funktionen. Zum Schluss folgen, für den Fall einer Weiterentwicklung, essenzielle und optionale Verbesserungen. Die Kontaktverwaltungs-App wurde als Projekt im Rahmen des Studienfaches Mobile Anwendungen bei Herrn Prof. Barth im Wintersemester 2020/2021 entwickelt.

Inhaltsverzeichnis

1 Einleitung

In der COVID-19-Pandemie sind Veranstaltungen mit mehreren Teilnehmern mit einem erhöhten Infektionsrisiko verbunden, gerade in geschlossenen Räumen. Häufig besteht an die Veranstalter eine Pflicht Ein- und Austrittszeitpunkt aller Teilnehmer individuell festzuhalten, um im Falle einer nachträglich auftretenden Infektion die Kontaktverfolgung für die Gesundheitsämter möglichst effizient zu gestalten. Eine Verwaltung mit Papierformularen ist mit einem hohen Digitalisierungsaufwand verbunden. Die App Kontaktverfolgung versucht dieses Problem zu lösen.

In der entwickelten App kann eine Nutzerin oder ein Nutzer die Rolle eines Hosts, also eines Gastgebers oder eines Teilnehmers einnehmen. Jeder Nutzer hat ein Profil bestehend aus: Name, E-Mail und Telefonnummer. Hosts erstellen Events die an eine reale Adresse gebunden sind. Teilnehmer können dann per NFC-Tag oder einen QR Code in dieses Event eintreten. Ein- und Austrittszeitpunkte werden erfasst und in der Datenbank vermerkt. Die Ein- und Austrittszeitpunkte der Events können als PDF exportiert werden. Eine ausführliche Demonstration der Funktionen ist auf Youtube zu finden.

2 Allgemeiner Aufbau der App

Im Wesentlichen besteht der Aufbau der App aus den in Abbildung 1 dargestellten Activites, dem RoomLifecycleService, dem MQTT-Service, einer SQLite Datenbank sowie den Preferences. Sämtliche benötigten Informationen über den Nutzer der App werden mittels shared Preferences erfasst. Ohne das vollständige Ausfüllen der Preferences ist ein Verwenden der App nicht möglich. Um später einen eleganten Zugriff auf Nutzerdaten in den Preferences zu erhalten ist eine *MySelf* Klasse implementiert. In den Activities werden den verwendeten Widgets ihre Funktionen zugewiesen. Sie bilden somit das Userinterface. Für den Datenaustausch zwischen Host und Teilnehmer kommt das Protokoll MQTT zum Einsatz. Dabei registrieren sich die Apps auf den Endgeräten auf eigene Informationsübertragungskanäle, die Topics genannt werden. Die Kommunikation wird schlussendlich durch den MQTT Service realisiert. Verändert sich der Status eines Events oder eines Teilnehmers, werden über den MQTT-Service die Datenbanken der anderen Teilnehmer/Host aktualisiert. Hierbei übernimmt die Datenbank die Speicherung von Nutzerdaten und von den Eventinformationen. Die Teilnahme an einem Event kann per NFC-Tag oder dem Abscannen eines QR-Codes erfolgen. Wird ein Room-Tag erkannt, werden die hinterlegten Daten aus der Klasse *MySelf* gelesen über den MQTT-Service an die Datenbank des Host übermittelt. Genauere Informationen über den Room-Tag sind in Abschnitt 4 zu finden. Events werden durch den RoomLifecycleService überwacht. Der RoomLifecycleService enthält einen Timer welcher den Lebenszyklus eines Events steuert.

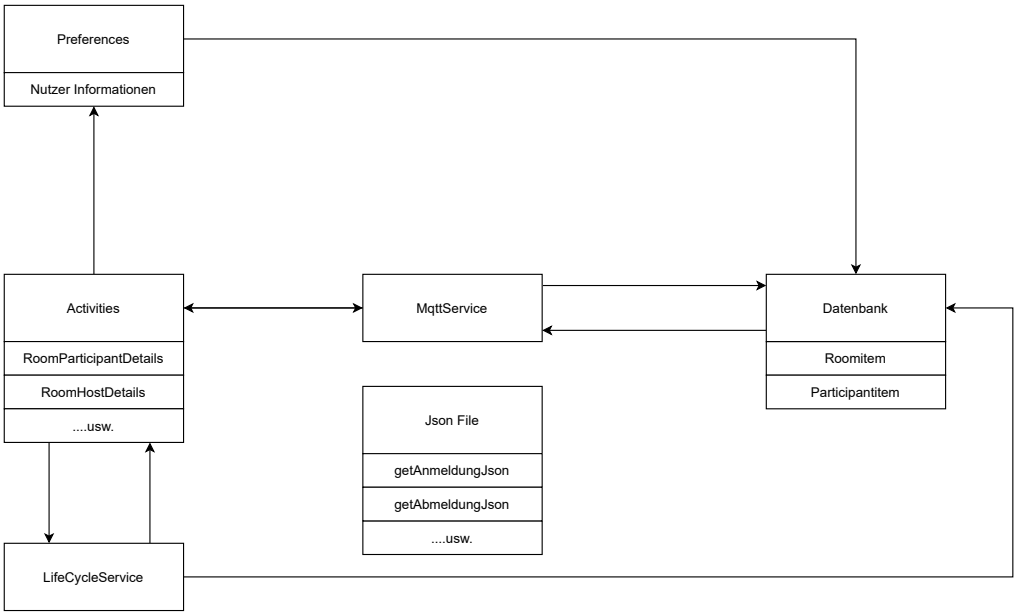

Abbildung 1: Grundlegendes Domänenmodel der entwickelten App.

Ändert sich der Lebenszyklus des Events, wird die Datenbank aktualisiert und die entsprechenden Teilnehmer über den MQTT-Service benachrichtigt. Die genaueren Funktionsweisen des RoomLifecycleService, MQTT-Server und SQL-Lite Da-tenbank werden in den folgenden Kapiteln erläutert.

3 Speicherung der Nutzerdaten

Die Datenbank wird als Speichermedium verwendet, um langfristig Information über Nutzdaten zu halten. In der Applikation werden die Tabellenstrukturen durch die Klassen *ParticipantItem* und *RoomItem* umgesetzt.
ParticipantItem ist eine Datenklasse und für die Verwaltung der Nutzerinformationen von Teilnehmern und Host zuständig. Tabelle 1 zeigt die Struktur der entsprechenden Datenbanktabelle *dbParticipant*. Neben dem Name und Kontaktinformationen wie E-Mail und Telefon enthält ein *ParticipantItem* Informationen über ein besuchtes Event. Die Attribute RoomId und Entertime erhalten beim Betreten eines Events einen fixen Wert. Die RoomId ist ein Fremdschlüssel auf ein *RoomItem*in der *dbRoom*. Kommt es nun zur Schließung eines Events, können mittels RoomId die dazugehörigen Teilnehmer ermittelt werden. Diese erhalten anschließend eine Exittime.
Die Tabellenstruktur der Events wird über die *RoomItem* Klasse realisiert. Der Aufbau der Tabelle *dbRoom* ist in Tabelle 2 dargestellt. Beim Erzeugen eines Events sind der Eventname, Name des Veranstalters, Ort des Events, Zeitraum und Status Pflichtangaben. Änderungen des Status werden über den in Abschnitt 4 näher beschrieben RoomLivecycleService durchgeführt. Da Hosts auch selbst Teilnehmer in einem ande-

Tabelle 1: Struktur der Datenbanktabelle dbParticipant

id	name	extra	eMail	phone	roomId	enterTime	exitTime
1	Max Mustermann	null	mm@mail.de	123456789	1	0	0

Tabelle 2: Struktur der Datenbanktabelle dbRoom

id	fremdId	roomName	status	host	eMail	phone	place	address	extra	enterTime	exitTime
1	null	MusterRaum	3 (ROOMISCLOSE)	Max Mustermann	mm@mail.de	+123456789	12345 Musterstadt	Musterstr. 1	null	0	0

ren Event sein können, wird hierfür das Feld der FremdID eingeführt. Ist das Attribut bei einer *RoomItem* Instanz initialisiert, wurde dieses Event von einer anderen Person erstellt und man wird als Teilnehmer gelistet. Die FremdID hält die RoomId aus der Datenbank des Hosts, diese wird benötigt um den Raum später noch mittels des Room-Tags Identifizieren zu können.
Um zu verhindern, dass Datenmanipulationen und Zugriffe nicht im Hauptthread stattfinden, sind im Repository (Klasse welche Funktionen des Data Access Objects implementiert) alle Datenbankaufrufe über seperat gestartete Threads implementiert. Die Ausnahme bilden die Funktionen mit der Endung *Now*. Da bei diesen keine LiveData zurückgeben wird dürfen diese nur für nebenläufige Threads genutzt werden. Endet eine Funktion mit dem Suffix *Own*, werden nur Informationen von eigens erstellte Events zurückgegeben. Dies wird vor allem im RoomLifecycleService verwendet damit Teilnehmer nicht selbst Events öffnen oder schließen können. Der Host allein regelt die Öffnungs- und Schließzeiten seiner Events. Eine genauere Dokumentation der einzelnen Funktionen ist im JavaDoc zu finden.

4 Steuerung des Eventlebenszyklus

Der RoomLifecycleService steuert, wie der Name bereits andeutet, in welchem Zustand ein Event befinden. Der Zustand bestimmt sich maßgeblich nach Öffnungs und Schließzeit. Diese sind abhängig von der lokalen Zeit. Ein Event soll ab einer gewissen Uhrzeit geöffnet und zu einer gewissen Uhrzeit geschlossen werden. Er hat drei mögliche Zustände:

- ROOMWILLOPEN
- ROOMISOPEN
- ROOMISCLOSE

Der RoomLifecycleService überprüft jede Sekunde, ob Events die geschlossen sind geöffnet werden sollen, ob Events die sich öffnen werden geöffnet werden können und ob Events die geöffnet sind geschlossen werden sollten. Er wird sowohl im Host, als auch im Teilnehmer-Modus benötigt. Die Entscheidung fiel auf einen Service, weil die App auch im nicht-geöffneten Zustand auf Timeouts und Öffnungszeiten reagieren

sollte. Als Beispiel folgender Problemfall: Der Host eröffnet einen Event und wechselt die App oder sperrt das Smartphone. Wenn die Kontaktverwaltungsapp nun nicht mehr im Vordergrund ist während das Timeout des Events ausläuft oder das Event in die Öffnungszeit reinrutscht, dann würde er nicht geschlossen respektive geöffnet werden. Um also zu vermeiden, dass der Host über die gesamte Zeit sein Smartphone nicht verwenden kann und das Display immer angeschaltet bleiben muss, was mit einem hohen Stromverbrauch verbunden ist, fiel die Entscheidung auf einen Service.

Wird der Service gestartet, wird die Raumüberprüfungsroutine nebenläufig in einem neuen Thread gestartet. Das nimmt die Last vom Hauptthread. Der Eventstatus wird jede Sekunde mit einem Busy-Wait-Algorithmus überprüft. Das sollte nicht in den Hauptthread ausgelagert werden, da er diesen unnötig blockiert. Der Datenbankzugriff ist also nicht die Ursache für die Designentscheidung, denn der Zugriff wird im Repository ohnehin nebenläufig realisiert. Die Überprüfung erfolgt also wie beschrieben in einem seperaten Thread. In diesem wird überprüft ob aktuell geschlossene Events geöffnet und aktuell geöffnete Events geschlossen werden sollten. Nebenher wird im Mainthread versucht den MQTT-Service zu einzubinden. Denn der Host soll natürlich auf die MQTT-Topics aller Event hören die offen sind bzw. aufhören auf alle Events zu hören die geschlossen sind. Außerdem muss der Host den Teilnehmern die Eventeigenschaften und den Öffnungsstatus beim Öffnen mitteilen. Der MQTT-Service wird hier also gebraucht um den Teilnehmern Event-Informationen mitzuteilen. Dabei ruft der RoomLifecycleService die Methoden des MQTT-Services nicht direkt auf. Da die Überprüfung nebenläufig erfolgt, kann es vorkommen, dass der Thread schon über den MQTT-Service Eventänderungen versendet obwohl dieser eigentlich noch nicht gebunden ist. In dem Fall kommt es zu einer Exception. Deshalb haben wurde ein Handler eingeführt. Der Thread schreibt in eine threadsichere Liste die Events rein, die ihren Zustand geändert haben und über MQTT versendet werden sollen. Ist der MQTT-Service im Hauptthread gebunden, dann arbeitet besagter Handler mit der Methode *postPendingRooms()* diese Events aus der threadsicheren Liste im Hauptthread ab. Dadurch ist stets sichergestellt, dass der MQTT-Service nur Events erhält wenn er gebunden ist. Dieser Workaround wäre vermeidbar gewesen, wäre Statusüberprüfung und Nachrichtenübertragung in einem zentralen Service realisiert worden. So hätte man gar nicht auf das Binden warten müssen.

5 Kommunikation über MQTT

Der MQTT-Service stellt die MQTT-Funktionalität bereit, über welche die App mit anderen Smartphones kommuniziert. Das Starten und Stoppen des MQTT-Service findet in der *Activity_00_StartActivity* statt. Alle anderen Activities binden lediglich den MQTT-Service. Das Binden des MQTT-Service findet in *Activity_11_EnterViaNfcQr*, *Activity_14_RoomParticipantsDetails* und *Activity_22_RoomHostDetails* statt. Des Weiteren wird der Service im RoomLivecycle-Service gebunden. Näheres hierzu unter

Abschnitt 4. Beim Verwenden des MQTT-Service wird in jeder Activity geprüft, ob dieser schon gebunden ist (not null). Ist dies nicht der Fall, wird das zu sendende Objekt zwischengespeichert und bei erfolgreichem Binden verarbeitet. Die im MQTT-Service implementierten Methoden sind ausführlich mittels JavaDoc erklärt.

Die Kommunikation zwischen dem Host und den Teilnehmern verläuft beim MQTT-Protokoll über sogenannte Topics. Für ein Event werden, wie Abbildung 2 zeigt, immer zwei Topics benötigt. Der Teilnehmer kann einem Event über die Topic *moagm/Room-Tag* beitreten. Der Host sendet die nötigen Informationen für die Teilnehmer über die Topic *moagm/Room-Tag/public* an alle Teilnehmer. Durch die Trennung der Kanäle wird verhindert, dass Teilnehmer die Nachrichten empfangen können, welche von anderen Teilnehmern an den Host gesendet werden.

Insgesamt gibt es vier verschiedene Nachrichtentypen, welche zwischen den Teilnehmern und dem Host ausgetauscht werden. Alle Nachrichten werden im JSON-Format versendet. Dabei lassen sich die Nachrichten in zwei Gruppen unterteilen. Die Nachrichten *anmeldung.json* und *abmeldung.json* werden vom Teilnehmer an den Host gesendet. *teilnehmer.json* und *rauminfo.json* sendet der Host an alle Teilnehmer. Mittels eines Konverters werden die entsprechenden JSON-Objekte erzeugt und wieder in verwendbare Datentypen zurück gewandelt. Zum Identifizieren eines Nachrichtentyps wird (warum auch immer..) nicht das Typfeld benutzt. Das Identifizieren wird mit den folgenden Schlüsselworten durchgeführt.

- ENTERTIME (*anmeldung.json*)
- EXITTIME (*abmeldung.json*)
- RAUM (*rauminfo.json*)
- TEILNEHMERLIST (*teilnehmer.json*)

Abhängig von dem empfangen Datentyp werden unterschiedliche Aktionen durchgeführt.
Mit der in Listing 1 dargestellten *anmeldung.json* meldet sich ein Teilnehmer beim Host an und stellt eine Verbindung zu dem entsprechenden Topic her. In der Nachricht sind alle relevanten Informationen des Teilnehmers enthalten sowie die Uhrzeit wann er den Event betreten hat. Informationen über einen Teilnehmer können den Preferences über die *MySelf* Klasse entnommen werden. Der MQTT-Service konvertiert die erhaltene JSON-Nachricht mittels der *AdapterJsonMqtt* Klasse in den passenden Datentyp. Das erhalte *ParticipantItem* Objekt wird in der Datenbanktabelle *dbParticipant* mit der entsprechenden RaumId abgespeichert. Die RaumId kann dem Topic entnommen werden, über welche die JSON-Nachricht empfangen wurde. Im Anschluss wird an alle Teilnehmer ein *teilnehmer.json* sowie *rauminfo.json* gesendet.

Die Nachricht *abmeldung.json*, dargestellt in Listing 2, sendet der Teilnehmer an den Host, sobald er das Event verlässt. Dabei meldet sich der Teilnehmer ebenfalls von

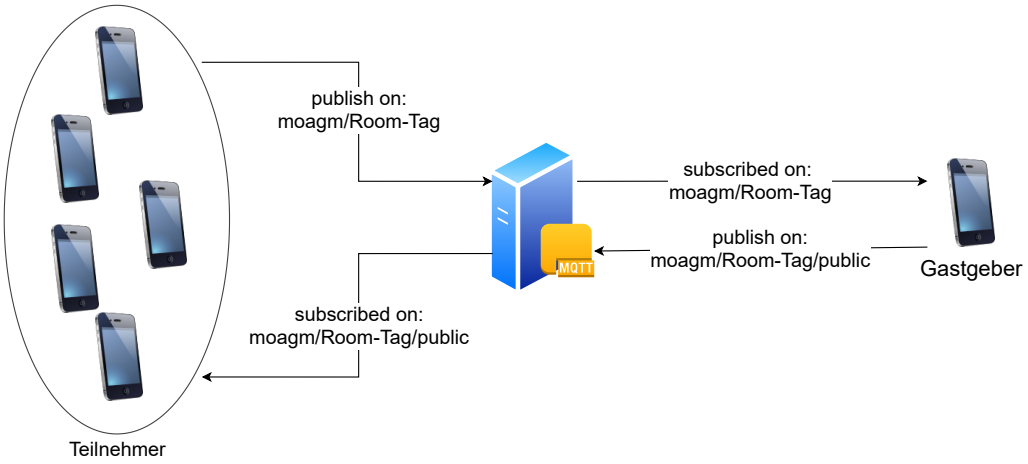

Abbildung 2: Darstellung des grundlegenden Kommunikationsablaufes mit MQTT-Topics.

der entsprechen Topic der MQTT-Verbindung ab. Analog zum *anmelden.json* enthält diese Nachricht alle wichtigen Informationen über den Teilnehmer sowie die Uhrzeit zu welcher er das Event um verlässt. Im MQTT-Service wird mithilfe des erzeugten *ParticipantItem* Objekt und der RaumId die Exittime des dazugehörenden Teilnehmers aktualisiert.
Listing 3 stellt die Nachricht vom Typ *rauminfo.json* da. Diese Nachricht enthält sämtliche Informationen über einen Event. Jede vom Teilnehmer empfange *rauminfo.json* wird in ein *RoomItem* Objekt konvertiert und in die Datenbanktabelle *dbRoom* eingefügt bzw. aktualisiert den bereits vorhanden Eintrag. Wenn der Host eine *rauminfo.json* versendet, welche den Teilnehmern mitteilt, dass das Events nun geschlossen ist, meldet der MQTT-Service den Host automatisch von der entsprechenden MQTT-Topic ab.
Der in Listing 4 dargestellte Nachrichtentyp *teilnehmer.json* beinhaltet eine Liste aller Teilnehmer eines Events. Bei Empfangen der Nachricht wird das darin enthaltene JSON-Array in eine Liste von *ParticipantItems* konvertiert. Die *ParticipantItems* der Liste werden mit der entsprechenden RaumId, welche über die Topic ermittelbar ist, in die Datenbanktabelle *dbParticipants* eingefügt. Um zu verhindern, dass Teilnehmer mehrfach eingetragen werden, wird die Anzahl der bereits eingetragenen Teilnehmer des Events ermittelt. Aus der Liste der erhaltenen Teilnehmer werden nur *ParticipantItems* eingefügt, welche einen mindestens gleich großen Index besitzen wie die ermittelte Anzahl. Diese Variante funktioniert, da sich die Reihenfolge der Teilnehmer in den Datenbanktabellen von Host und Teilnehmer 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},
        ...
 ]
}
```

6 Betreten eines Events über QR-Code und NFC

Damit Events distinktiv sind besitzen Sie einen Identifier. Dieser nennt sich Room-Tag und ist zusammengesetzt aus dem roomName, der eMail des Hosts und der roomId aus der Datenbank des Hosts. Beispiel für einen Room-Tags: *Vorlesung 1/dozent@hs-mannheim.de/1*. Teilnehmer benötigen diesen Tag um dem Event beizutreten. Dies können Sie über NFC oder über einen QR-Code durchführen. Eine manuelle Eingabe ist nicht implementiert.

Um den Room-Tag weiterzugeben kann der Veranstalter nach dem Erstellen des Events ein PDF generieren und optional per E-Mail versenden. Das passiert in der Methode *shareRoom()* in der Klasse *Activity_22_RoomHostDetail*. In einem solchen

PDF ist neben Event-Metadaten wie Name, Start und End-Zeitpunkt auch ein QR-Code über den die Teilnehmer beitreten können. Für die Erstellung wird die externe Bibliothek zxing von journeyapps benutzt. Die Anbindung erfolgt über die Klasse *BarcodeEncoder* in der Klasse *QRCodeManager*.

Ein Schreiben des Room-Tags auf einen NFC Sticker ist nicht implemenentiert. Apps wie NFC Tools des Unternehmens wakdev aus dem PlayStore schaffen hier aber Abhilfe. Dort kann der Room-Tag im Plaintext-Format auf den NFC-Tag geschrieben werden. Eine URI soll nicht vergeben werden, da der NFC-Scanner nicht drauf reagieren wird. Im Laufe der Entwicklung wurde fest definiert, dass ein Veranstalter seine eigenen Events nicht nochmal als Teilnehmer betreten dürfte. Außerdem darf ein Teilnehmer, nachdem er eine Sitzung verlassen hatte, nicht wieder erneut an der Veranstaltung teilnehmen. Damit wurden irreführende Einträge in der Auswertung vermieden. Dieses Verhalten wird mit der Methode *checkEnterPeriomission()* in der *Activity_11_EnterViaQrNfc* abgefangen.
Wünscht der Teilnehmer über einen QR-Code einzutreten, dann wird in der *Activity_11_EnterViaQrNfc* die Funktion *callScanner()* aus der *QrCodeManager* Klasse gerufen. Dort wird die Scanner-Activitiy aus zxing gestartet, die einen Rückgabewert liefert. Es ist auch möglich den Scanner vorzukonfigurieren, so, dass z.B. immer der Kamerablitz angeschaltet ist um auch bei schlechten Licht-Bedingungen den QR-Code gut zu erkennen. Außerdem kann ein Timeout eingestellt werden, falls der Nutzer den QR-Code in einer gewissen Zeit nicht einscannt. In unserem Projekt wurde sich aber wegen der Benutzerfreundlichkeit gegen diese Konfiguration entschieden. Der Scanner wird über einen Intent gestartet und liefert wenn er fertig ist ein Ergebnis zurück. Dieses wird in der Methode *onActivityResult()* verarbeitet. Wenn der Scan geklappt hat wird die Methode *enterRoom()* aufgerufen. Dort werden die Daten des Room-Tags in die Datenbank eingetragen und die *Activity_14_RoomParticipantDetail* gestartet. Der Teilnehmer hört dort dann auf das korrekte MQTT-Topic. Er hat das Event betreten.
Wünscht der Teilnehmer über NFC einzutreten, dann wird in der *Activity_11_EnterViaQrNfc* die Funktion *armNFCAdapter()* aufgerufen. Es wird überprüft ob das Smartphone überhaupt einen NFC Sensor hat. Falls das erfüllt ist wird dieser auf events des Types ACTION_NDEF_DISCOVERED scharfgeschaltet. Man sagt auch er ist im Foreground-Dispatch-Modus. Das bedeutet dass, wenn ein neues NFC-Tag detektiert wurde, ein gewisser Intent gestartet wird. Falls das der Fall ist, wird die gleiche *Activity_11_EnterViaQrNfc* noch einmal gestartet allerdings mit der Action NFC_INTENT_ACTION. In der *onResume()* wird überprüft ob diese Action gesetzt ist, denn nur dann kehrt die Anwendung aus dem NFC-Intent zurück. In den Extras des NFC-Intents befindet sich dann er abgelesene Room-Tag. Der Ablauf danach ist wie beim Betreten über einen QR-Code. *enterRoom()* wird gerufen und *Activity_14_RoomParticipantDetail* wird gestartet.

7 Erstellen und Teilen von PDFs

Das System soll dem Nutzer die Möglichkeit bieten Informationen über ein Event als PDF zu teilen. Dabei gibt es zwei unterschiedliche PDF‘s, diese beinhalten die generellen Eventinformationen, sowie den dazugehörigen QR-Code oder eine Liste aller Teilnehmer. Das Android-Framework verfügt über eine *PdfDocument* Klasse, die zum Erstellen von PDF‘s genutzt wird. Für eine A4 Seite nach DIN Norm ergeben sich die Werte 846x594 Pixel. Näheres zur Berechnung der Pixelwerte ist unter developer.android zu finden. Die erstellte Seite wird mittels eines Canvas editiert. Die Formatierung der Texte und Grafikelemente lassen sich über die *Paint* Klasse einstellen. Das erstellte PDF-Dokument wird im externen Dateiverzeichnis 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 eine Auswahl der installierten Anwendungen, über welche er das PDF teilen kann.

8 Weitere Funktionalitäten

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

9 Fazit und Ausblick

Die Anwendung ermöglicht eine effiziente, einfache Möglichkeit dezentral und mit wenig Ressourcen eine Kontaktverfolgung umzusetzen. Sie erfüllt die Grundsätze der Datenminimierung, Zweckbindung, Richtigkeit und Speicherbegrenzung der Datenschutz Grundverordnung. Nutzerdaten werden nur auf den Endgeräten gespeichert. Sie werden nur gespeichert, sollte ein Teilnehmer oder ein Host auch wirklich Kontakt mit diesem Nutzer gehabt haben. Außerdem werden alle Einträge die älter als 14 Tage sind aus der Datenbank gelöscht.

Für eine Weiterentwicklung der App sind jedoch essenzielle Verbesserungen vorzunehmen. Die Services MQTT-Service und RoomLivecycleService müssen in einem einzigen Service zusammengefasst werden. Dadurch lässt sich die Fehlerrate, welche beim Binden des MQTT-Services entstehen deutlich reduzieren. Auf etwaige Handler

könnte verzichtet werden. Das Zusammenfassen verringert zudem deutlich die Gesamtkomplexität des Projektes. Ebenso muss in der *PdfClass* das PDF mit der Übersicht aller Teilnehmer überarbeitet werden. In der aktuellen Version wird die E-Mail Adresse ab einer bestimmten Länge zum Teil mit in die nächste Spalte geschrieben. Ziel ist es, dass in dieser Situation ein automatischer Zeilenumbruch in der entsprechenden Spalte eingefügt wird. Neben den essenziellen Verbesserungen wäre eine Implementierung von folgenden Features erwägenswert.

- Beschreiben von NFC-Tags in der App ohne Drittanbieter-App
- Eigenes URI-Format für NFC-Tags
- Manuelle Eingabe als Möglichkeit einen Event zu betreten.
- Profilbild für die Nutzer
- Benachrichtigen aller Teilnehmer bei positivem Corona-Test
- Chat zwischen Teilnehmern eines Events
- Unittests unter Verwendung der *TestClass* implementieren

Eine verbesserte Version der Kontaktverfolgung App lässt sich ideal in der Hochschule einsetzen. Gerade für die Veranstaltung von Präsenzvorlesungen bietet die App die nötige Funktionalität um die Teilnahme von Studenten zurückzuverfolgen. Im Falle einer Infektion kann so die Zusammenarbeit mit den Gesundheitsämtern effektiver vonstattengehen.