Hochschule Anhalt
Anhalt University of Applied Sciences

Projektarbeit

Kontaktverfolgung mit einem Qr-Code-Scanner

Eingereicht von:	Annika Piechutta, Marcus Gagelmann, Charlotte Hintzen
Matrikelnummer:	5040979, 5041161 , 5041145
Studienfach:	Interaktive Mediensysteme
Betreuender Hochschullehrer:	Michael Cebulla
Tag der Einreichung:	08. Februar 2022

Selbstständigkeitserklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Halle, 20. Januar 2022

Charlotte Hintzen, Annika Piechutta, Marcus Gagelmann

Kurzfassung

In dieser Arbeit wird eine Kontaktverfolgungs-Applikation mit Hilfe eines QR-Code-Scanners prototypisch in Android Studio implementiert. Mit der Applikation ist es möglich einen QR-Code eines Ortes einzuscannen, den Zeitraum des Aufenthalts, den Namen des Nutzers und den Ort in einer Datenbank zu speichern und anschließend die Begegnungen in einer Liste auszugeben.

Inhaltsverzeichnis

1 Motivation und Einleitung

Die vorliegende Arbeit befasst sich mit der Erstellung einer Kontaktverfolgungs-Applikation. In diesem Kapitel wird die Motivation und Zielsetzung dieser Arbeit erläutert. Qr-Codes finden sich in unserem Alltag sehr häufig und werden immer beliebter. Sie haben eine Vielzahl von Anwendungsmöglichkeiten. Sie enthalten Informationen, die nach bestimmten Vorgaben verschlüsselt und anschließend codiert dargestellt werden. Außerdem spielt die Kontaktverfolgung aufgrund der Corona-Pandemie eine immer wichtigere Rolle.

Ziel dieser Projektarbeit im Rahmen des Moduls *Entwicklung Mobiler Anwendungen* ist es eine Kontaktverfolgungs-App, in Anlehnung an die *Luca-App* [1], prototypisch zu implementieren. Damit sollen später Kontakte nachvollzogen und entstandene Infektionsketten ermittelt werden können.

2 Konzeption und Implementierung

In diesem Kapitel werden die Anforderungen, die Umsetzung der einzelnen Schritte sowie die Probleme näher erläutert.

2.1 Anforderungsanalyse

Die Idee unserer Applikation ist eine Kontaktverfolgung mittels QR-Code. Nutzende können sich per QR-Code für bestimmte Orte und Zeiten registrieren. Für die Benutzenden kann mittels eines entsprechenden Algorithmus eine Liste aller Begegnungen erstellt werden. Hierbei ist zu erwähnen, dass diese prototypische Implementierung die Richtlinien des Datenschutzes nicht berücksichtigt.

2.1.1 Funktionale Anforderungen

Funktionale Anforderungen legen fest, was das Produkt tun soll und welche Komponenten dafür benötigt werden. Die funktionalen Anforderungen für das Projekt sind folgende:

- **Qr-Code Scanner** erfasst den Qr-Code des Ortes
- Nutzer kann Name in einem **Anmeldeformular** eingeben
- Liste, gibt die **Begegnungen** aus
- **Ändern** des Namen nachträglich

2.1.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen gehen über die funktionalen Anforderungen hinaus. Sie beschreiben, wie gut das System die Leistung erbringen soll. Folgende nicht-funktionale Anforderungen wurden für dieses Projekt festgelegt:

- Erkennen der Qr-Codes
- Eine **Liste** speichert den Namen des Benutzers nach der ersten Eingabe, sodass der Name nicht erneut eingegeben werden muss
- Die **DB** speichert die Daten, wie z.B. Uhrzeit, Name, Ort des Nutzers

2.2 Umsetzung

Die Umsetzung lässt sich in die Schritte Grundaufbau der Applikation, Login-Speichern, QR-Code Scanner, ExpandableListView, Swipe-Funktion und Datenbank-Back-End unterteilen. Diese werden in diesem Kapitel detailliert beschrieben.

2.2.1 Grundaufbau der Applikation

Die Android Applikation besteht insgesamt aus drei Activities, sechs Layouts und einem Menü-Layout. Öffnet sich die Applikation, erscheint bei dem erstmaligen Öffnen der Applikation ein Pop-up. Hier müssen sich die Nutzenden mit Vor- und Nachnamen anmelden. Jedoch darf der Name keine Sonderzeichen beinhalten. Sonst wird eine Fehlermeldung ausgegeben. Wird der Namen erfolgreich eingegeben, öffnet sich zunächst ein Warndialogfeld. Hier muss der Applikation die Erlaubnis erteilt werden, auf die Kamera zugreifen zu dürfen. Wurde diese Zugriffsberechtigung erteilt, öffnet sich der Qr-Code Scanner (*MainActivity*). Hier gibt es die Möglichkeit, einen QR-Code einzuscannen. Des Weiteren kann, unter dem Menüpunkt „Daten ändern“, welchen sich oben rechts in der Ecke befindet, der Name geändert und gespeichert werden. Wird von rechts unten nach oben links geswiped landet man auf der *SwipeLeftActivity*. Hier sehen die Nutzenden alle ihre Orte, an denen sie sich eingescannt haben. Öffnen sie einen Unterpunkt, kann der oder die Nutzende ebenfalls andere Nutzenden der Applikation sehen, die diesen Ort innerhalb von zwei Stunden vor und zwei Stunden nach ihm besucht haben.

2.2.2 Login-Speichern

Um den eingegebenen Namen zu speichern werden *Shared Preferences* verwendet. Diese bieten die Möglichkeit, kleine Mengen primitiver Daten als Schlüssel/Wert-Paare in einer Datei auf dem Gerätespeicher zu speichern und abzurufen. Sie bilden ihre Einstellungen in einer XML-Datei innerhalb der Applikation auf dem Gerätespeicher. Zunächst wird eine Einstellungsdatei für die Anwendung erstellt. Anschließend werden die gespeicherten Daten daraus geladen und in den Eingabefeldern abgebildet. Dies ist in Abbildung 1 zu sehen.

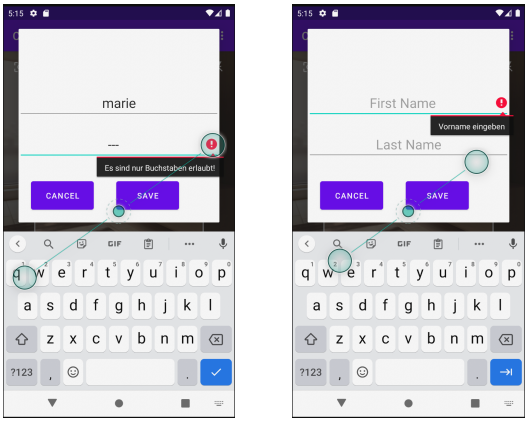
```
1        SharedPreferences mySPR = getSharedPreferences("Pref", 0);
2
3        newcontactpopup_firstname.setText(mySPR.getString("vornameKey",
             ""));
4        newcontactpopup_lastname.setText(mySPR.getString("nachnameKey",
             ""));
```

Abbildung 1: Shared Preferences

Sobald der „Save-Button“ geklickt wird, wird die Eingabe auf ihre Korrektheit überprüft. Ist sie korrekt wird der Name in die Datei gespeichert. Andernfalls lässt sich der Name nicht speichern und es erscheint eine Fehlermeldung wie in Abbildung 3 zusehen ist. Die Funktionen, welche sowohl beim ersten Login als auch in dem Menü unter dem Punkt „Name ändern“ zu finden sind, sind in dem Code-Abschnitt in Abbildung 2 zu sehen.

```
1         button_save.setOnClickListener(new View.OnClickListener(){
2             @Override
3             public void onClick(View v) {
4                     String n = newcontactpopup_firstname.getText().
                          toString();
5                     String ph = newcontactpopup_lastname.getText().
                          toString();
6
7                     if(newcontactpopup_firstname.length()==0){
8                         newcontactpopup_firstname.setError("Vorname␣
                              eingeben");
9                     }
10
11                    else if(!n.matches("[a-zA-z]+")){
12
13                        newcontactpopup_firstname.setError(
14                        "Es␣sind␣nur␣Buchstaben␣erlaubt!");
15                    }
16
17                    else if(newcontactpopup_lastname.length()==0){
18                        newcontactpopup_lastname.setError("Nachname␣
                              eingebn");
19                    }
20
21                    else if(!ph.matches("[a-zA-z]+")){
22
23                        newcontactpopup_lastname.setError(
24                        "Es␣sind␣nur␣Buchstaben␣erlaubt!");
25                    }
26                    else{
27
28                        SharedPreferences mySPR = getSharedPreferences("
                              Pref",
29                        0);
30
31                        SharedPreferences.Editor editor = mySPR.edit();
32
33                        editor.putString("vornameKey", n);
34                        editor.putString("nachnameKey", ph);
35
36                        editor.commit();
37
38                        dialog.dismiss();}
39
40                }
41        });
```

Abbildung 2: Funktionen für die Fehlermeldungen beim Login



(a) Fehlermeldung bei falschen Zeichen

(b) Fehlermeldung bei leerem Feld

Abbildung 3: Screenshots der Login-Page mit Fehlermeldungen

2.2.3 QR-Codes erstellen

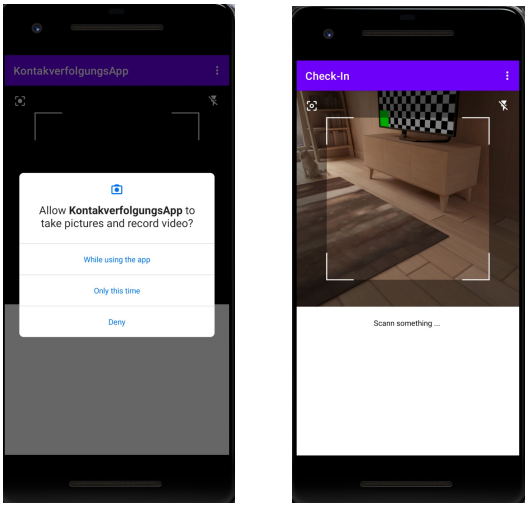
Für die verschiedenen Orte werden Qr-Codes zum scannen benötigt. Diese wurden mit Hilfe eines Qr-Code-Gernerators online erstellt. Diese Qr-Codes geben eine Zahl aus, welche einem Ort in der Datenbank zugeordnet ist. Dies wird in dem Unterabschnitt 2.2.7 näher erläutert.

2.2.4 QR-Code Scanner

Für die Erstellung des QR-Code Scanner haben wir eine Library basierend auf ZXing [2] verwendet. In diesem Kapitel werden die einzelnen Schritte bei der Erstellung des Scanners erläutert.

Kamera Zugriffsberechtigung

Wird die Applikation geöffnet, startet die *MainActivity*, auf welcher der QR-Code Scanner dargestellt ist. Bevor ein QR-Code einscannt werden kann, muss der Zugriff auf die Kamera akzeptiert werden, wie in Abbildung 4a zu sehen ist. Dies wurde mittels der Systemberechtigung *„android.permission.CAMERA“* im *Manifest* umgesetzt. Sobald der Zugriff auf die Kamera gestatte wurde, öffnet sich der QR-Code Scanner. Dies ist in Abbildung Abbildung 4b abgebildet. Um die Kameraberechtigung zu erhalten, wird ein Warndialogfeld verwendet. Dieses Feld besteht aus einem Titel, einer Nachricht und einem positiven und einem negativen Button. Der positive Button wird verwendet um weiterzumachen, in diesem Fall den Zugriff auf die Kamera zu erlauben. Der negative Button dient zum Abbrechen der Aktion, also den Zugriff zu verweigern.



(a) Screenshot der Kamera Zugriffsberechtigung

(b) Screenshot des QR-Code-Scanners

Abbildung 4: Verschiedene Screenshots der Applikation

QR-Code scannen

Wurde die Erlaubnis auf die Kamera erteilt, kann ein QR-Code gescannt werden. Wird von dem QR-Code Scanner ein QR-Code erkannt, wird dieser eingelesen. Bei erfolgreichem Scannen wird eine kleine Textnachricht angezeigt, dass das Scannen erfolgreich war. Wenn ein QR-Code eingelesen wird, wird ein neuer Besucher in der Datenbank angelegt. Hierbei werden Name, *DateTime* und der Ort an die Datenbank übergeben und gespeichert.

2.2.5 ExpandableListView

Wird in der *MainActivity* von unten rechts nach oben links gewischt gelangt man zu der *SwipeLeftActivity*. Hier können die Begegnungen eingesehen werden. Zunächst wird eine Liste der besuchten Orte aufgelistet. Klickt man auf einen Ort, öffnet sich ein Untermenü. Hier können andere Besucher, die den gleichen Ort besucht haben, angezeigt. Die Begegnungen werden mittels einer *ExpandableListView* ausgegeben. Dies ist eine Ansicht, die Elemente in einer vertikalen scrollenden zweistufigen Liste anzeigt. Die Elemente stammen aus dem *MainAdapter*, der mit dieser Ansicht verbunden ist.

2.2.6 Swipe-Funktion

Öffnen die Nutzenden die Applikation, erscheint der QR-Code Scanner. Die Benutzenden befindet sich zu diesem Zeitpunkt auf der *MainActivity*. Um auf die *SwipeLeftActivity* zu gelangen, auf der die Begegnungen abgebildet sind, muss von unten rechts nach oben links gewischt werden. Möchte man nun wieder zu dem QR-Code Scanner zurück, muss auf die

von Androide vor implementierte Zurück-Taste geklickt werden. Um die Touch-Geste zu ermöglichen, müssen zunächst die Daten über ein Touch-Ereignis gesammelt werden. Danach müssen die Daten interpretiert werden, um festzustellen, ob sie die Kriterien für die Swipe-Funktion erfüllt. Wird ein Finger auf dem Bildschirm platziert, löst dies den *Callback onTouchEvent()* aus. Die Geste beginnt, wenn die Nutzenden den Bildschirm das erste Mal berühren, setzt sich fort, wenn das System den Finger verfolgt, und endet mit dem Erfassen des letzten Ereignisses, wenn der Finger den Bildschirm verlässt. Während dieser gesamten Interaktion liefert das *MotionEvent*, welches an *onTouchEvent()* übergeben wird, die Details jeder Interaktion. Der Code dazu ist zu Veranschaulichung in Abbildung 5 dargestellt.

```
1  @Override
2     public boolean onTouchEvent(MotionEvent event)
3     {
4         switch(event.getAction())
5         {
6             case MotionEvent.ACTION_DOWN:
7                 x1 = event.getX();
8                 break;
9             case MotionEvent.ACTION_UP:
10                x2 = event.getX();
11                float deltaX = x2 - x1;
12                if (Math.abs(deltaX) > MIN_DISTANCE)
13                {
14                    // Right to Left swipe action
15                    if (x2 < x1)
16                    {
17                        Intent i = new Intent(Activity_Swipe_Left.this,
                              MainActivity.class);
18                        startActivity(i);
19                    }
20
21                }
22
23                break;
24        }
25        return super.onTouchEvent(event);
26    }
```

Abbildung 5: Java-Code für die Swipe-Funktion

2.2.7 Datenbank-Back-End

Im Betrieb der Kontaktverfolgungsanwendung müssen in Echtzeit die Besuchszeitpunkte von verschiedenen Personen an unterschiedlichen Orten zum Abgleich abgespeichert und ausgelesen werden können. Zur Umsetzung dieser Funktionalität war die Implementierung

eines zentralen Datenbankservers für eine realitätsnahe Umsetzung unserer Idee notwendig. Im folgenden wird dieses Datenbank-Server-System unseres Projektes näher erläutert.

Anforderungen

Im Regelbetrieb unserer App soll bei erstmaligem Start pflichtmäßig ein neuer Nutzer angelegt werden. Außerdem sollen Nutzer zu einem späteren Zeitpunkt die Möglichkeit erhalten ihren Namen zu ändern. Hierfür sollte das Datenbanksystem Funktionalitäten zum Erstellen, Speichern und Anpassen von Nutzern bereitstellen. Weiterhin sollten Orte im System vorliegen können, an denen Nutzer die Möglichkeit erhalten, sich für einen gewissen Zeitraum zu registrieren. Hierfür wird eine Ortstabelle innerhalb der Datenbank benötigt, welche sowohl den Namen eines Ortes als auch seinen regulären Besuchzeitraum eindeutig einer QR-Code-ID zuordnet. Zusätzlich sollte es Clients mithilfe des Servers möglich sein, den Besuch von Orten in einer weiteren Tabelle zusammen mit einem konkreten Zeitpunkt festzuhalten. Um alle Besuche eines Nutzers als Liste innerhalb der App abbilden zu können, sollten Clients eine Funktion erhalten, welche die Menge aller Besuche eines Nutzers aus der Datenbank ausliest. Abschließend ist zu berücksichtigen, dass unsere App Nutzern helfen soll, ihre Kontakte zu anderen Personen an bestimmten Orten und zu konkreten Zeitpunkten nachzuvollziehen. Zu diesem Zweck ist die Implementierung einer Funktionalität im Datenbanksystem sinnvoll, die unter Angabe eines Ortes und eines Zeitpunktes die Menge alle Namen von Nutzern ausgibt, welche sich zur gleichen Zeit am gleichen Ort befunden haben.

Aufbau

Das Datenbank-Server-System, welches an die App angebunden wurde, besteht aus einer *PostgreSQL-Datenbank*, welche auf dem Server-Rechner mit einer Java-Anwendung über die *java.sql* Bibliothek kommuniziert. Diese Java-Serveranwendung hingegen nimmt Anfragen vom Client über Sockets per TCP/IP entgegen und beantwortet diese auf gleiche Weise.
In Abbildung 6 wird das Entity-Relationship-Modell der verwendeten PostgreSQL-Datenbank dargestellt. Erkennbar sind die Tabellen *Users*, *Places* und *Visits*. Primärschlüssel werden in dieser Darstellung mit *PK* (*Primary Key*) und Fremdschlüssel mit *FK* (*Foreign Key*) gekennzeichnet. Die Users-Tabelle stellt die Nutzerverwaltung des Datenbanksystems dar und erstellt zu abgespeicherten Nutzernamen automatisch eine Nutzer-ID, die sogenannte UID. Ähnlich verhält es sich mit der Ortsverwaltung und der Places-Tabelle. Hier wird lediglich noch ein Zeitfenster zu jedem Ort abgespeichert, welches angibt, wie lange ein Nutzer an diesem Ort registriert wird. Zuletzt ist in der Abbildung die Visits-Tabelle zu erkennen. Eine Zeile dieser Tabelle repräsentiert den Besuch eines Nutzers an einem bestimmten Ort. Ein solcher Besuch besteht aus einer Nutzer-ID, einer Orts-ID und einem konkreter Zeitpunkt vom Datentyp *DateTime*.

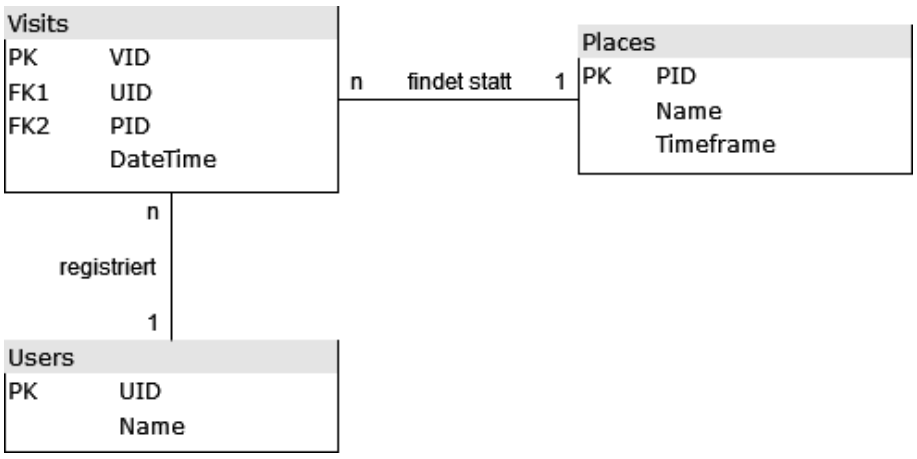


Abbildung 6: Entity Relationship Modell der PostgreSQL-Datenbank
Quelle: Eigene Darstellung

Im laufenden Anwendungsbetrieb ist es der Clientanwendung jederzeit möglich, Anfragen an den Server mithilfe des in Abbildung 7 dargestellten Interfaces zu senden. Die fünf dargestellten Funktionen sind bereits stark auf die Anforderungen an das Back-End abgestimmt und erfüllen die im folgenden genannten Zwecke:

1. Die Funktion *scanQR* sendet eine UID, eine PID und einen Zeitpunkt an den Server, womit ein neuer Besuch im system registriert wird, und gibt anschließend den zum Ort zugehörigen Zeitrahmen zurück.
2. Die Funktion *newUser* nimmt einen Nutzernamen entgegen, legt damit im Back-End einen neuen Nutzer in der Datenbank an und liefert einschließend die UID dieses neuen Nutzers.
3. Die Funktion *setName* ändert den Namen eines bestehenden Nutzers unter Angabe seiner UID und seines neuen Namens.
4. Die Funktion *loadVisits* benötigt die UID eines Nutzers und liefert anschließend aus dem Back-End eine Liste von allen Besuchen, welche für diesen Nutzer in der Datenbank registriert sind.
5. Die Funktion *loadMetPeople* nimmt eine PID und einen Zeitpunkt entgegen. Diese werden vom Server genutzt, um ein Array aller Namen von Personen zu liefern, welche zu dieser Zeit an diesem Ort gewesen sind.

Das Senden von Anfragen vom Client an den Server erfolgt mithilfe der Klasse *MessageSender*, welche die Klasse *AsyncTask* erweitert und welche in Abbildung 8 dargestellt wird. Hiermit wird für jede einzelne Anfrage ein asynchroner Task erstellt, welcher den Server-Socket-Code für das senden einer TCP/IP-Nachricht an den Server ausführt. Um die Ant-

```
1  public interface Communication {
2          public static double scanQR(int UID, int PID, String DateTime) {
3                  return 0;
4          }
5
6          public static int newUser(String NewUser) {
7                  return 0;
8          }
9
10         public static void setName(int UID, String NewName) {
11         }
12
13         public static ArrayList<Visit> loadVisits(int UID) {
14                 return null;
15         }
16
17         public static String[] loadMetPeople(int PID, String DateTime) {
18                 return null;
19         }
20 }
```

Abbildung 7: Interface der von der Applikation nutzbaren Datenbank-Client-Funktionen
Quelle: Eigene Darstellung

worten des Servers empfangen zu können, wird mit dem Start der der App auch ein zusätzlicher Thread gestartet. Die Aufgabe dieses Threads ist es, sämtliche TCP/IP-Antworten, welche vom Server geschickt werden, zu empfangen und an die zugehörigen Funktionen des Client-Interfaces (siehe Abbildung 7) weiterzuleiten.

2.3 Aufgetretene Probleme

Während der Implementierung sind einige Probleme aufgetreten, welche in diesem Kapitel aufgezählt werden.

Shared Preferences In der Applikation ist es an verschiedenen Stellen möglich den Namen zu ändern. Es hat zunächst nur das Ändern am Anfangs-Login funktioniert. Nach einigen Versuchen hat es mit der in dieser Dokumentation gezeigten Lösung funktioniert.

Swipe-Funktion Die Nutzenden sollten die Möglichkeit haben, mittels Wischen über den Bildschirm vor und zurück in der Applikation zugehen und somit die Ansichten (Activitys) mit Hilfe einer Gesten-Funktion zu verändern. Das *onTouchEvent* funktionierte auf der *MainActivity* ohne Probleme. Jedoch war es nicht möglich, von den Begegnungen wieder zu dem QR-Code Scanner zuswipen. Dies sollte mit dem wischen von unten links nach oben

```
1  public static class MessageSender extends AsyncTask<String,Void,Void> {
2          Socket s;
3          DataOutputStream dos;
4          PrintWriter pw;
5          String ip = "";
6          int port = 0;
7
8          MessageSender(String ip, int port) {
9              this.ip = ip;
10             this.port = port;
11         }
12
13         protected Void doInBackground(String... voids) {
14             String message = voids[0];
15             try {
16                 s = new Socket(ip, port);
17                 pw = new PrintWriter(s.getOutputStream());
18                 pw.write(message);
19                 pw.flush();
20                 pw.close();
21                 s.close();
22             } catch (IOException e) {
23                 e.printStackTrace();
24             }
25             return null;
26         }
27     }
```

Abbildung 8: Die Klasse *MessageSender*, welche die Klasse *AsyncTask* erweitert
Quelle: Eigene Darstellung

rechts möglich sein. Jedoch funktionierte dies nicht. Auch nach reichlicher Recherche konnte keine hilfreiche Information gefunden werden. In jedem Tutorial oder in jeder Hilfestellung wurde von der zweiten Activity mit der von Android vor implementierte Zurück-Taste auf die erste Activity gewechselt. Erst bei der Recherche über Back Stacks konnte herausgefunden werden, dass eine Rücknavigation von Android nur über die Zurück-Taste möglich ist. Es gibt die Möglichkeit, eine benutzerdefinierte Rücknavigation zu implementieren, jedoch wurde sich aufgrund des Zeitaufwands dagegen entschieden. Auf Grund dessen wurde die Rücknavigation per Swipe verworfen und auf die vor implementierte Zurück-Taste von Android zurückgegriffen.

Einbindung des Client-Codes Der Server- sowie der Client-Code für die Kommunikation mit der Datenbank wurde, abseits der Android-App, in einer Java-IDE entwickelt. Nach Fertigstellung des Kommunikationsprogramms war dessen Einbindung in die App über eine vorher festgelegte Schnittstelle (siehe Abbildung 7) notwendig. Hierbei stellte sich heraus,

dass Sockets für die TCP/IP-Kommunikation im Code einer Android-App die Verwendung asynchroner Programmierkonzepte erfordern. Mit dieser Erkenntnis ging ebenfalls die Aufgabe der Aneignung von Kenntnissen über die Programmierung von Threads in Java einher, um den fertig geglaubten Client-Code umzubauen und korrekt in die App einpflegen zu können. Die Aufgabe der Einbindung des Client-Codes erforderte aus diesem Grund einen wesentlich höheren Arbeitsaufwand als ursprünglich angedacht war.

Netzwerkkommunikation Im Rahmen dieses Projektes war es nicht möglich, einen geeigneten Server mit benötigter statischer öffentlicher IP-Adresse zu erhalten. Aus Zeitgründen und aufgrund der beschränkten Mittel wurde innerhalb des Projekts ein Server-Prototyp entwickelt, welcher lediglich mit einem Gerät gleichzeitig kommunizieren kann und welcher auf die Verwendung in lokalen Netzwerke mit privaten IP-Adressen abgestimmt ist. Diese Lösung wurde vom Team als ausreichend bewertet, da der Fokus des Projektes auf einer funktionierenden und präsentierbaren Android-Applikation liegt und hierdurch alle Funktionen der App wie beabsichtigt verwendet werden können. Sämtliche Tests und Präsentationen der Funktionen der App wurden unter Verwendung eines privaten Rechners eines Gruppenmitglieds als Server durchgeführt.

2.4 Fazit

Das Ziel des Projekts wurde in allen Punkten erreicht: Die Kontaktverfolgungs-App konnte erfolgreich in Android Studio implementiert werden, wie in diesem Video zu sehen ist.

Abbildungsverzeichnis

Literaturverzeichnis

[1] ANONYM. *Luca-App*. URL: https://www.luca-app.de, besucht am 28. 01. 2022,

[2] ANONYM. *Code scanner library*. URL: https://github.com/yuriy-budiyev/code-scanner#readme, besucht am 17. 01. 2022,