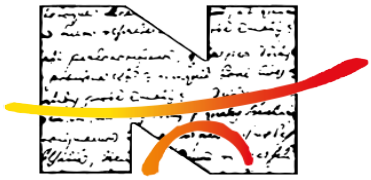
Gymnasium Nepomucenum Coesfeld
Besondere Lernleistung im Abitur 2022



DMergency - App zur Alarmierung und Verwaltung von Sanitätsdiensten

Entwicklung einer App zur effizienten und benutzerfreundlichen Alarmierung, sowie Verwaltung von Sanitätsdiensten

vorgelegt von

Mattis Rinke

Fachbereich Informatik

Herr Brumma
Herr Willenbring

Inhaltsverzeichnis

1 Einleitung **4**

2 Das Projekt **5**

2.1 Zielsetzung 5

2.2 Abgrenzung zur Server Ausarbeitung 6

3 Wahl der Entwicklungsweise **6**

3.1 Version 1 - Native Entwicklung 7

3.2 Version 2 - Xamarin Forms 8

3.3 Version 3 - Flutter 9

4 Entwicklung **10**

4.1 Funktionen der App 10

4.1.1 Rollen und Registrierung 10

4.1.2 Alarmauslösung 15

4.1.3 Alarmempfang 17

4.1.4 Vertretungen 18

4.1.5 News & Notfallnummern 19

4.1.6 Berechtigungen & Fehler 19

4.2 Kommunikation mit dem Server 20

4.2.1 API-Nutzung 21

4.2.2 Nutzung von Firebase-Messaging 21

4.3 Speicherung der Daten 25

4.3.1 Umsetzung 25

4.3.2 Aufbau der Datenbank 26

5 Schultests **29**

6 Fazit **31**

6.1 Was wurde erreicht 31

6.2 Wie geht es weiter 31

7 Abbildungsverzeichnis **32**

8 Literaturverzeichnis **32**

9 Anhang **34**

1 Einleitung

Der Drang nach Digitalisierung wird in der heutigen Welt immer größer, so auch bei (Schul-)Sanitätsdiensten. Diese Notwendigkeit und das Bedürfnis nach vereinfachten Abläufen habe ich selbst erfahren und ist mir durch meine leitende Tätigkeit im Schulsanitätsdienst besonders deutlich geworden, da ich hier Diensteinteilungen, Vertretungen erkrankter Mitschüler/-innen und die Alarmierung mit den Funkgeräten organisieren muss, wodurch ich selbst die Digitalisierung von Sanitätsdiensten als sinnvoll und notwendig erachte. Hier wurde bisher meist mit Funkgeräten oder auch mit Schuldurchsagen alarmiert, was den Schulunterricht drastisch gestört hat. Außerdem ist die Alarmierung selbst sehr ineffizient, da zunächst eine Person im Sekretariat oder ähnlichem informiert werden muss, die permanent als Ansprechpartner vor Ort sein muss, informiert wird, um im Anschluss daran die Sanitäter/-innen zu alarmieren.

Deshalb suchte ich nach Apps, die eine Entlastung sowie eine einfache Alarmierung ermöglichen.Ich stieß auf mehrere Apps bei der Recherche, unter anderem sie Apps SSanialarmünd "Divera 24/7". Beide weisen grundsätzlich die Voraussetzungen zur Nutzung im Schulsanitätsdienst auf: Beide Apps können Alarmieren und einen Dienstplan erstellen. Jedoch haben auch beide Einschränkungen, wodurch diese nicht komplett für den Schulsanitätsdienst geeignet sind, wie sich durch die unzuverlässige Alarmierung bei Sanialarm und die umständliche Dienstplan-Erstellung bei Divera, bei der man jeden Tag (also pro Datum) einen Dienst erstellen muss, feststellen ließ, so dass an dieser Stelle keine tatsächliche Erleichterung durch die Digitalisierung erfolgt.

Durch meine Vorkenntnisse im Fach Informatik bin ich dann schnell auf die Idee gekommen selbst eine App zu selbst zu gestalten und habe es mir, zusammen mit einem Freund, zur Aufgabe gemacht, diese Probleme zu lösen sowie reibungslose Abläufe und Planungen für einen Schulsanitätsdienst zu ermöglichen. In dieser Ausarbeitung gehe ich auf den Entwicklungsprozess der App ein, warum ich mich für das Framework Flutter entschieden habe,

welche Funktionen die App hat und wie diese umgesetzt wurden.

Als erstes werde ich skizzieren, was die App können soll, weshalb bestimmte Anwendungsvoraussetzungen von Bedeutung sind und beschreibe im Anschluss, wie ich mich von der nativen Entwicklung für die Nutzung des Cross-Platform-Frameworks Flutter entschieden habe. Danach erläutere ich detaillierter die einzelnen Funktionen der App, woraufhin die Kommunikation mit dem Server und die Speicherung der Daten näher erörtert wird. Zuletzt führe ich noch Ergebnisse der ersten Schultests auf und ziehe dann ein Fazit, in dem ich ausführe, was in der Ausarbeitung bereits umgesetzt wurde und wie die weitere Entwicklung sowie der Umgang mit der App erfolgen kann und soll.

2 Das Projekt

2.1 Zielsetzung

Die App soll das Alarmieren und Verwalten von Sanitätsdiensten vereinfachen. Um dies zu verwirklichen ist es erforderlich mehrere Funktionen zu implementieren. Zum einen ist eine Funktion zum Alarmieren unerlässlich, welche zur Vergewisserung für die alarmierende Person auch ein Feedback anzeigen sollte, zum anderen sollte es dann auch eine Funktion zum Empfangen des Alarms geben. Diese beiden Funktionen sollten so implementiert werden, dass eine alarmierende Person so wenig Aufwand wie möglich in der Durchführung der Alarmierung hat, die Sanitäter/-innen jedoch trotzdem so viele Informationen wie möglich bekommen. Damit eine Verständigung seitens der Sanitäter/-innen darüber erzielt werden kann, wer sich für das Einsatzmaterial verantwortlich zeigt, ist es erforderlich, dass dies ebenfalls mit in die Funktionen aufgenommen wird. Darüber hinaus ist die Programmierung eines Dienstplan-Systems vorgesehen, das nur diensthabende Sanitäter/-innen alarmiert. Selbstverständlich gibt es im (Schul-)Alltag auch Situationen, in denen Sanitäter/-innen kurzfristig (z.B. im Falle von Krank-

heiten) oder auch absehbar längerfristig (z.B. durch angekündigte Arbeiten, Ausflüge oder Praktika) ihren Dienst nicht durchführen können, so dass es eine Funktion geben muss, mit der sich diese austragen können und andere ihren Dienst übernehmen. Im Notfall ist eine schnelle Rettungskette von großer Bedeutung. Um diese Hilfskette [1] einzuhalten, sollen in der App wichtige Notfallnummern hinterlegt werden, welche dann durch einen schnellen Klick wählbar sind.

Ein weiterer wichtiger Bestandteil zur Verwaltung des Sanitätsdienstes ist die Kommunikation zwischen der Leitung und den Mitgliedern des Sanitätsdienstes. Um diese Kommunikation sicherzustellen soll eine News-Funktion programmiert werden, in welcher die Mitglieder Neuigkeiten von der Leitung einsehen können. Hierzu ist es unabdingbar eine Funktion umzusetzen, die es der Leitung ermöglicht, News schreibenßu können.

2.2 Abgrenzung zur Server Ausarbeitung

In dieser Ausarbeitung wird die Funktionsweise der App „DMergency"“ beschrieben und wie sie in Zusammenarbeit mit dem Server arbeitet. Es wird nicht darauf eingegangen, wie der Server arbeitet und welche Funktionen es in der Web-Anwendung gibt. Zum Teil werden Daten vom Server verarbeitet oder auf diesem gespeichert. In diesen Fällen wird dies erwähnt jedoch nicht weiter auf die Verarbeitung eingegangen.

3 Wahl der Entwicklungsweise

Es gibt in der Programmierung etliche Möglichkeiten der Programmierung. Auch in der Entwicklung für mobile Endgeräte. Mit dem Projektstart wählte ich den mir zunächst am sinnvollsten erscheinenden Weg der nativen Entwicklung. Die native Entwicklung. Dadurch, dass die App für unterschiedliche Betriebssysteme erhältlich sein soll, muss die native Entwicklung sowohl für das Betriebssystem iOS [3] von Apple durchgeführt werden, als auch für das Be-

triebssystem Android [2], von Google. Es gibt aber auch die Möglichkeit der Cross-Platform-Programmierung, bei der für beide Betriebssysteme gleichzeitig programmiert wird. Wie schon erwähnt, entschied ich mich zunächst für die native Programmierung, jedoch bewegte ich mich im Entwicklungsprozess von der nativen Entwicklung zur Cross-Platform- Programmierung mit dem Framework Xamarin-Forms, um schließlich das Framework Flutter zu verwenden. Worin die Vor- und Nachteile liegen und warum ich mich letztendlich für die Cross-Platform-Programmierung mit Flutter entschied wird in den nächsten drei Unterkapiteln erläutert.

3.1 Version 1 - Native Entwicklung

Der Einstieg in die native Entwicklung erfolgte mit Android, da hier für mich eine garantierte Kompatibilität mit dem Betriebssystem vorlag. Diese stellte sich als mühelos heraus, da in der Programmiersprache Java geschrieben wird, die mir aus dem Informatik- Unterricht bereits bekannt war. Nach kurzer Zeit lag eine vorläufig fertig programmierte App vor, die einen zuverlässigen Empfang und das Auslösen eines Alarms ermöglichte. Da dieser App jedoch auch zuerst nur für den Schulsanitätsdienst meiner eigenen Schule gedacht war gab es hier kein Registrier- und Login-Verfahren genauso wenig wie die Notfallnummern, die News und die Einstellungen. Diese App war also nur auf den Sanitätsdienst meiner eigenen Schule zugeschnitten.

Da es jedoch auch iOS-Geräte in diesem Sanitätsdienst gibt, musste ich schnell merken, dass ich auch hier eine App programmieren muss. Also habe ich mich informiert und habe wie ich für iOS entwickle und die App auch für iOS-Geräte veröffentlichen kann. Die Programmierung für iOS findet nativ in den Programmiersprachen Swift und Objective-C statt, was mich vor ein weiteres Problem gestellt hat. Ich bin weder mit der Programmiersprache Swift noch mit der Programmiersprache Objective-C vertraut, was bedeuten würde, dass ich mir diese von null auf beibringen müsste. Dies wäre zwar sehr unangenehm, jedoch möglich gewesen. Ich habe jedoch erst ein Mal

weiter recherchiert und bin auf die nächste Barriere gestoßen: Um die App veröffentlichen zu können muss ich einen Apple-Developer-Account haben um die App im Appstore veröffentlichen zu können. Da diese 99€ im Jahr kostet bin ich zu dem Entschluss gekommen, dass es für mich nicht möglich ist die App nur für den Sanitätsdienst meiner eigenen Schule zu entwickeln. Um dieses Problem zu umgehen habe ich mich dazu entschieden die App nicht nur für meinen eigenen Sanitätsdienst zu entwickeln, sondern diese auch an weitere Sanitätsdienste zu verkaufen. Um dies zu bewerkstelligen musste ich jetzt also zum einen die App sowohl für Android, als auch für iOS zu entwickeln, da der Marktanteil von Apple bei 30% und der von Google bei 70% liegen

Da der Markt zwischen Apple und Google in Sachen Handy-Betriebssystemen bei ca. 70% zu 30% liegt[4, vgl.], habe ich schnell gemerkt, dass ich die App auch für iOS entwickeln muss. UM sicherzustellen, dass alle Funktionen immer auf jeder Plattform zur Verfügung stehen, ist es bei dieser Entwicklungsmethode notwendig, jede Funktion zweimal zu programmieren. Dies ist als Einzelperson nicht machbar, weshalb ich mich neu orientierte. Recherchearbeiten führten mich dann zu der Methode des Cross-Platform-Programmings, welche in den nächsten zwei Unterkapiteln erläutert werden.

3.2 Version 2 - Xamarin Forms

Xamarin Forms ist ein Framework der **.NET-Plattform** von Microsoft. Dieses Framework ist ein Cross-Platform-Framework, das heißt, dass der Code einmalig für die beiden Betriebssysteme (Android & iOS) geschrieben wird und die App dann für beide erhältlich ist. Xamarin Forms hat eine Unterteilung zwischen dem funktionalen Code, welcher in C# geschrieben ist, und zwischen der der Markup Language XAML, welche das GUI darstellt[5, vgl.]. Jedoch sind bei der Nutzung von Cross- Platform-Frameworks auch Einbußungen zu machen. In diesem Fall konnte ich mich zum einen nicht mit der Markup Language XAML anfreunden, da hier das neu laden des

GUIs als sehr sperrig und aufwendig herausstellte, zum anderen stellte sich heraus, dass Xamarin einige von mir benötigte Funktionen nicht voll oder gar nicht unterstützt. Unter anderem gab es immer wieder Probleme beim Einbinden von Firebase-Messaging, ein Tool von Google, zum Versenden von Push-Notifications (Auf Firebase-Messaging wird im weiteren Verlauf der Ausarbeitung noch eingegangen). Durch diese für mich nicht lösbaren Probleme musste ich mich dann erneut auf die Suche nach einer anderen Lösung machen. Um diese Lösung geht es jetzt im nächsten Abschnitt.

3.3 Version 3 - Flutter

Die dritte und bis jetzt finale Version ist in Flutter geschrieben und die am weitesten entwickelte App-Version. Die Entscheidung für Flutter viel nach mehreren Empfehlungen, z.B. durch einen Freund, welcher bereits durch ein Praktikum bei der Firma d.velop mit diesem Erfahrung sammeln konnte, sowie nach eigener Recherche. Flutter nutzt eine einfache Programmiersprache, die Java ähnelt, weshalb mir der Einstieg in die Programmiersprache Dart[6] nicht schwergefallen ist. Flutter ist wie Xamarin- ein Cross-Platform Framework. Dieses ist in der Lage Apps für die beiden gängigen Plattformen iOS und Android zu kompilieren[1], sowie für das Web. Das Nutzer-Interface ist in Flutter durch so genannte Widgets, welche in einem Widget- Tree aufgenommen werden, dargestellt. Hierbei wird dann zwischen S̈tateless"[8]- und S̈tateful"[9]-Widgets unterschieden. Der Unterschied liegt darin, dass in S̈tateless-Widgetsßwar Variablen, etc. abgeändert werden können, jedoch wird das Widget im GUI nicht neu gerendert. Es können aber in dem Widget selbst andere Widgets, die selbst S̈tateful-Widgetsßind, aktualisiert werden. Ein Stateful-Widget besteht aus zwei Klassen. Der Klasse, die um die Klasse S̈tateful-Widgetërweitert wird und der Klasse, die um die Klasse State[10] erweitert wird. In der Klasse mit dem Stateful Widget wird in der c̈reateState-

[1]Kompilieren beschreibt das Umwandeln des geschriebenen Programmtextes in ein funktionsfähiges Programm

Methode die Klasse mit dem State zurückgegeben. Der eigentliche Widget-Tree wird dann erst in der ̈build-Methode"der State-Klasse geschrieben und dann auch zurückgegeben. Nach der Erstellung des Widget-Trees kann bei Bedarf durch die Methode s̈etState"das UI aktualisiert werden. Außerdem gibt es asynchrone Methoden und Futures[11]. Diese sind dazu da das Programm weiterhin Code ausführen zu lassen, während das Programm gleichzeitig darauf wartet, dass die asynchrone Methode fertig wird. Dies wird oft dazu genutzt Daten vom Netzwerk zu laden oder in eine Datenbank zu schreiben. Die meist genutzten Datentypen sind: int, String, bool, List, Future und Map. Nach einiger Überlegung und Recherchen ist die Entscheidung letztendlich für flutter als Framework gefallen, weil zum einen nur einmal Code für die beiden Plattformen iOS und Android geschrieben werden muss und zum anderen das GUI einfacher programmierbar ist als z.B. mit Xamarin Forms und Flutter sehr performant ist.

4 Entwicklung

4.1 Funktionen der App

In den folgenden Abschnitten werden jetzt die Funktionen der App dargestellt und erklärt. Dazu werden beispielhaft einzelne Methodenimplementationen herausgenommen, erörtert und im Kontext der jeweiligen Funktion erklärt. Außerdem werden Design-Entscheidungen skizziert um die Nutzer-Erfahrung zu visualisieren.

4.1.1 Rollen und Registrierung

Wie bereits in der Zielsetzung angesprochen, soll es ein Berechtigungssystem für die App geben. Dieses wird unterteilt in die Rollen Sanitäter/-in und Alarmierende/r. Alarmierende sollen nur in der Lage sein einen Alarm auszulösen und die Alarmierenden-News, sowie die Notfallnummern einzusehen.

Anders als die Alarmierenden sollen Sanitäter/-innen auch in der Lage sein einen Alarm zu empfangen und andere Sanitäter/-innen zu vertreten oder sich aus dem Dienstplan auszutragen. Um dies umzusetzen muss bereits bei der Registrierung darauf geachtet werden, wer welche Rolle zugewiesen bekommt. Im Folgenden erkläre ich jetzt, zunächst beispielhaft, wie die Registrierung für einen Sanitäter abläuft und erkläre im Anschluss, welche Unterschiede es bei der Registrierung für einen Alarmierenden gibt.

Zunächst muss der/die Nutzer/-in den ihr/ihm zugehörigen Sanitätsdienst auswählen. Um den Sanitätsdienst auszuwählen muss der / die Nutzer/-in auf den gewünschten Sanitätsdienst wählen. Wenn dieser nicht direkt zu finden ist, ist es außerdem möglich diesen mit der Suchleiste oben zu suchen. Im Anschluss muss dann auf den "WeiterButton geklickt werden um sich im Anschluss anzumelden oder zu Registrieren. Der Button zeigt außerdem an, ob es möglich ist auf diesen zu klicken oder nicht, je nachdem ob bereits ein Sanitätsdienst ausgewählt worden ist oder nicht. Dies wird dadurch angezeigt, dass dieser grau ist, wenn der Button nicht angeklickt werden kann und grün, wenn er angeklickt werden kann.
In Abbildung 1 ist der View zu sehen, in welchem der Sanitätsdienst ausgewählt wird.

Um dies umzusetzen muss beim Start der App zunächst überprüft werden ob bereits ein Nutzer eingeloggt ist oder nicht.

1. Es muss überprüft werden ob schon ein Nutzer eingeloggt ist. In diesem Fall soll keine Sanitätsdienstauswahl angezeigt werden, sondern die nor- male Nutzeroberfläche, für die entsprechende Rolle.
2. Wenn bisher kein Nutzer eingeloggt ist muss die Sanitätsdienstauswahl angezeigt werden und die Sanitätsdienste vom Server geladen werden.

Nachdem der/die Sanitäter/-in dann seinen/ihren Sanitätsdienst ausgewählt hat, soll nun ein Login-View angezeigt werden, bei welchem man

sich entscheiden kann ob man:
1. sich Einloggen möchte.
2. sich neu Registrieren möchte.
3. sein Passwort vergessen hat und dieses zurücksetzen möchte.
In der Nutzeroberfläche ist dies so umgesetzt, dass je ein Eingabefeld für die E-Mail-Adresse und das Passwort angegeben ist. Unter diesem sind dann je zwei Texte, einmal Registrierenünd einmal Passwort vergessen", über welche dann auf die unterschiedlichen Views geleitet wird. Ganz unten in dem View ist dann erneut ein Button, welcher dann zum Anmelden genutzt werden kann (Auch hier gibt es wieder die Unterscheidung nutzbar/nicht nutzbar).
Im Code sieht das ganze dann so aus, dass in einem View „Validate " ein FutureBuilder zurückgegeben wird, der so lange einen LoadingIndicator angezeigt (Also einfach nur einen sich drehenden Kreis, der an einer Stelle eine Lücke hat) bis die „future" ausgeführt wurde. Die future ist eine Methode, welche async ist. In diesem Fall wird durch diese Methode die Rolle des/der Nutzer/-in bestimmt. Hierfür werden zunächst die Login-Daten aus der Datenbank gelesen und überprüft ob diese vorhanden sind. Danach wird überprüft, welche Rolle in der Datenbank steht und diese je nachdem passend für die API abgeändert (In der Datenbank ist 0 gespeichert, was für eine/-n Sanitäter/-in steht → Abänderung zu 3, da ein/-e Sanitäter/-in bei der API eine 3 ist). Danach wird durch eine Request an den Server geprüft ob der Nutzer noch existent ist, indem die Sanitätsdienst-ID, die Nutzer-ID und die Rolle als query-Teil mitgegeben. Danach wird überprüft, was die Antwort des Servers ist. Ist diese eine 0 wurde der Account gelöscht und der/die Nutzer/-in wird ausgeloggt und auf die Sanitätsdienstauswahl geleitet. Ist die Antwort eine 2 wird der/die Nutzer/-in auf den View der eigenen Rolle weitergeleitet, kann jedoch keine Alarme versenden, da der Account vorübergehend gesperrt ist. Bei diesen beiden Fällen wird jeweils ein AlertView angezeigt, der darüber informiert, was „falsch" gelaufen ist. Im Falle einer 1 als Antwort ist alles normal und der/die Nutzer/-in wird

auf den passenden View für die jeweilige Rolle weitergeleitet.

Um sich anzumelden muss man bereits einen existenten Account besitzen. Im Login-View muss man dann seine E-Mail-Adresse, die mit dem Account verknüpft ist, sowie das zugehörige Passwort angeben, um dann eingeloggt zu werden. Die Account-Daten werden nach der Login-Bestätigung durch den Nutzer zum Server geschickt und überprüft, um dem/der Nutzer/-in im Anschluss eine Fehler-Meldung anzuzeigen oder sie/ihn auf die Nutzeroberfläche weiterzuleiten.

Um sein Passwort zurückzusetzen muss man einen neuen View öffnen, bei welchem man seine E-Mail-Adresse angeben kann. Nachdem der/die Nutzer/-in diese bestätigt hat, wird eine E-Mail and die E-Mail-Adresse geschickt, durch die man dann sein Passwort zurücksetzen kann.

Das Registrier-Verfahren ist im Gegensatz zum Login und Passwort zurücksetzten komplizierter gestaltet. Nach dem Aufruf, dass man einen neuen Account erstellen möchte muss zunächst ein Passwort angegeben werden, dass durch den Sanitätsdienst festgelegt wurde. Dieses Passwort unterscheidet je nach Rolle die zugeteilt werden soll. Das heißt: Nutzer/-in A soll die Rolle Sanitäter/-in bekommen, als gibt er/sie Passwort „xyz“an. Nutzer/-in B soll die Rolle Alarmierende/-r bekommen, also gibt die Person Passwort „abc“an.
Nach der Verifizierung dieses Passworts geht es dann für die beiden Rolle unterschiedlich weiter.

1. Sanitäter/in
Zunächst müssen Sanitäter/-innen ihren Vor- und Nachnamen zur Identifizierung für die Sanitätsdienst-Leitung angeben. Außerdem muss eine Stufe (bestehend aus drei beliebigen Zeichen) und ein Geschlecht angeben. Diese Daten dienen der Sanitätsdienst-Leitung zur Identifizierung und Planung des Sanitätsdiensts.
Danach muss der/die Nutzer/-in eine E-Mail-Adresse und ein Passwort

angeben, um sich erneut anmelden zu können und den Nutzer im System eindeutig zu identifizieren. Das Passwort muss, um sicherzugehen, dass das Passwort angegeben wurde, was gewünscht ist, zweimal eingegeben werden. Im Anschluss daran werden dann App-Berechtigungen abgefragt. Warum es diese gibt und weshalb ich diese Berechtigungen abfrage, wird jetzt dargelegt. In den Betriebssystemen Android und iOS, gibt es zum einen Funktionen, die ohne jegliche weitere Berechtigungen genutzt werden können (Dies sind meist Funktionen, welche keine kritischen Nutzerdaten erfordern sondern nur zur App-Funktionalität beitragen. So zum Beispiel das Speichern von App-Daten). Jedoch stellen die beiden Herausgeber der jeweiligen Betriebssysteme (Google und Apple) auch Funktionen zur Verfügung, welche Berechtigungen benötigen, welche durch den Nutzer der App erlaubt werden müssen. Dies dient dem Schutz der Daten der Nutzer der Betriebssysteme. Ich habe mich entschlossen einige Berechtigungen abzufragen, damit ich den Nutzern einige Funktionen zur Verfügung stellen zu können. Auf Android-Geräten habe ich mich dazu entschlossen die Berechtigung den Nicht-Stören-Modus zu überschreiben abgefragt, damit ich jeder Zeit einen Alarm and die Sanitäter/-innen senden kann und auch einen Sound abspielen kann sollte sich das Gerät in einem Stumm-Modus oder dem nicht Stören-Modus befinden. Eine weitere Berechtigung auf Android-Geräten ist die Aufhebung der Batterie-Optimierung. Diese soll für die App deaktiviert werden, damit diese dauerhaft im Hintergrund laufen kann und die Alarme empfangen kann. Auf iOS-Geräten fordere ich zum einen an, dass ich Mitteilungen senden darf, da dies anders als bei Android eine Berechtigung erfordert, zum anderen frage ich die Critical-Alert-Berechtigung ab, damit ich genauso wie bei Android-Geräten den Stumm- und den Nicht-Stören-Modus überschreiben darf.

2. Alarmierende

Alarmierende müssen zunächst ähnlich wie die Sanitäter/-innen einen

Account-Namen festlegen. Dieser besteht jedoch nicht aus Vor- und Nachname sondern nur aus einem generalisierten Namen, da hier theoretisch auch Account-Namen für feste Räume eingetragen können, wie z.B. bei einem Schulsanitätsdienst das Sekretariat. Danach muss genauso, wie bei den Sanitäter/-innen eine E-Mail zur eindeutigen Identifikation eines Accounts angegeben werde, über welche auch das Passwort, welches im Anschluss angegeben werden muss, zurückgesetzt werden kann. Das Passwort muss genauso wie bei den Sanitäter/-innen zweimal eingegeben werden um die richtige Eingabe von diesem sicherzustellen. Im Anschluss muss jetzt nur noch durch das Betätigen des RegistrierenButtons die Registrierung abgeschlossen werden, durch das man dann auf das Nutzer-Interface weitergeleitet wird.

4.1.2 Alarmauslösung

Das Alarmauslösen, soll wie bereits erwähnt, möglichst einfach für die/den Alarmierende/-n sein, da diese meist Laien sind und sich dadurch in einer Alarmsituation so oder so schon in einer Ausnahmesituation befinden und es hier keine große Hürde sein sollte sich Hilfe zu besorgen.

Um dies zu bewerkstelligen sind auf dem Server Alarmorte vorgespeichert. Diese werden dann beim Aufruf des Views, auf dem der Alarm gesendet wird heruntergeladen und im Anschluss angezeigt. Um diese auszuwählen muss der/die Nutzer/-in zunächst einen Ort-Typen aus einem Dropdown-Menü auswählen, welcher dann im Anschluss entscheidet, was als genauer Ort angegeben wird. Die genauen Orte können entweder genau den gleichen Wert haben wie der Ort-Typ, oder eine Auswahl an Objekten, oder eine Zeicheneingabe, welche entweder eine Nummerneingabe oder eine Texteingabe sein kann. Diese werden dann entweder als Dropdown-Menü, Eingabefeld oder als nicht abänderbarer Text angezeigt. Zunächst muss die Alarmierende Person also den Ort-Typen spezifizieren, danach kann der Nutzer dann den genauen Ort spezifizieren. Danach kann der/die Nutzer/-in eine Beschreibung des

Geschehens verfassen, damit der/die Sanitäter/-in sich bereits vor dem Eintreffen an der Einsatzstelle ein Bild von der Lage machen kann, hierfür wird dann ein Eingabefeld angezeigt, in dem der/die Nutzer/-in die Beschreibung angeben kann. Zuletzt kann der/die Alarmierend/-e eine Priorität zwischen 1 und 5 festlegen um die Dringlichkeit klar zu machen. Hierbei ist 1 die niedrigste Priorität und 5 die höchste. Diese Prioritätsauswahl ist durch RadioButtons dargestellt.

Der letzte Schritt, der dann noch ausgeführt werden muss ist die Betätigung des „Alarm senden“-Buttons, welcher dauerhaft sichtbar am unteren Ende des Views angezeigt wird, um problemlos jederzeit alarmieren zu können.

Nachdem der Alarm gesendet wurde, was durch den Server geregelt wird, nachdem von der App aus eine Request hierfür gestellt wurde, wird der alarmierenden Person ein View angezeigt, auf welchem Rückverfolgt werden kann, welche/-r Sanitäter/-in den Alarm entweder empfangen, abgelehnt oder bestätigt hat. Wenn der Alarm noch nicht empfangen wurde wird, um dies zu symbolisieren ein Ladekreis angezeigt, wenn der Alarm empfangen wurde, wird ein schwarzer Hacken angezeigt, wenn der Alarm bestätigt wurde wird ein grüner Hacken angezeigt und wenn der Alarm abgelehnt wurde wird ein rotes Kreuz angezeigt.

Nachdem drücken des Alarm-Buttons wird zunächst die Methode sendAlarm(BuildContext context) aufgerufen. Im Anschluss werden dann zuerst die Login-Daten aus der Datenbank gelesen, um dies mit in die Alarmierungs-Request zu schreiben. Im Anschluss darauf werden dann die Beschreibung, die Priorität und der Alarmierungsort aus den jeweiligen Eingabefeldern ausgelesen und ebenfalls in der Request angegeben. Danach wird die Request gesendet und überprüft, ob es eine erfolgreiche Antwort des Servers gibt (also ob der http-Statuscode 200 ist) und ob die Antwort eine gültige AlarmID enthält. Wenn diese Bedingungen zutreffen wird dann auf den bereits erwähnten Informations-View weitergeleitet, dem die AlarmID des Ser-

verantwort mitgegeben wird. Sollte eine der Bedingungen nicht erfüllt sein wird wieder ein AlertDialog angezeigt, der den/die Nutzer/-in über den Fehler informiert. Danach wird der View neugebaut um die eingegeben Daten zurückzusetzen.

4.1.3 Alarmempfang

Um einen Alarm versenden zu können, muss dieser logischerweise auch empfangen werden können. Dies wird über das Cloud-Messaging-System, Firebase Cloud-Messaging, von Google, gelöst. Beim Empfangen des Alarms (also der Nachricht von Firebase Cloud-Messaging), wird ein Alarm-Sound abgespielt, damit der/die Sanitäter/-in diesen mitbekommt. Im Anschluss werden die Alarm-Daten jetzt in die Datenbank geschrieben. Wenn der/die Sanitäter/-in die App nun öffnet, kann diese/-r den View „Alarminformationen “öffnen, auf welchem die empfangenen Alarme angezeigt werden. Wenn der/die Nutzer/-in jetzt auf den neuesten Alarm klickt werden die Alarminformationen hier erneut, jedoch detaillierter angezeigt. Hier kann der/die Sanitäter/-in jetzt auch Rückmeldung auf den Alarm geben, in dem er/sie jetzt den Alarm durch die jeweiligen Buttons ablehnt oder annimmt. Die Daten werden in der Reihenfolge Alarmierungszeitpunkt, Ort, Beschreibung, Priorität angezeigt. Dies dient zum einen natürlich der Information des Sanitäters/ der Sanitäterin, zum Anderen soll so auch dem/der Sanitäter/-in gezeigt werden, wann der Alarm gesendet wurde um zu wissen, dass er/sie sich ggf. beeilen muss, da der Alarm, z.B. durch eine schlechte Internet-Verbindung erst verspätet angekommen ist. Nach der Alarmierungszeit ist dann der Ort wichtig, da der/die Sanitäter/-in sich dort möglichst schnell hinbewegen sollte und erst dann sind Beschreibung und Priorität wichtig, da dies nur Zusatzinformationen sind, die der/die Sanitäter/-in so oder so am Einsatzort erheben muss.

Die Funktion der Alarm-Rückmeldung ist zum einen zum sicherstellen, der Funktionsfähigkeit der App implementiert worden, zum anderen soll so

auch der alarmierenden Person Sicherheit gegeben werden, dass Hilfe kommt um sie in der ihr vorliegenden Ausnahmesituation, zu unterstützen / zu helfen. Die Alarmrückmeldung erfolgt über die Buttons am unteren Ende des Views. Hier sind zunächst die Buttons "Bestätigenünd Äblehnen"gegeben, im Anschluss wird dann entweder nur die Rückmeldung, die man selbst gegeben hat angezeigt, damit der/die Sanitäter/-in dies überprüfen kann, oder sollte man den Alarm bestätigt haben und noch niemand anderes durch die Betätigung des Buttons Material holen", gezeigt hat, dass dieser dies übernimmt, wird der Button Material holenängezeigt um Beschriebenes anzuzeigen. Wenn bereits von jemandem das Material geholt wird, wird dies in einem weiteren Feld angezeigt und anstatt der Buttons wird das Feedback für den Alarm angezeigt.

4.1.4 Vertretungen

Um die Sanitäter/-innen zu Alarmieren ist auf dem Server ein Dienstplan hinterlegt, welcher bestimmt wer an welchem Tag alarmiert wird. Jedoch kann es auch hier zu Ausnahmesituationen kommen, in denen die eigentlich diensthabende Person nicht in der Lage ist den Dienst zu verrichten. Damit hier schnell eine Lösung gefunden werden kann, ist es möglich sich in der App zum einen aus dem Dienst auszutragen, und zum Anderen kann man auch, sollte jemand einem Bescheid gegeben haben, dass die Person ihren Dienst nicht verrichten kann, eine andere Person temporär für diesen Tag vertreten. Dies wurde so umgesetzt, dass alle diensthabenden Personen in der App, in einer Liste angezeigt werden. Hier kann der Nutzer dann entweder eine der diensthabenden Personen vertreten, in dem sie diese auswählt und dann auf Vertreten klickt, oder sich selbst austragen, in dem die Person auf den „Austragen "Button klickt.

Die Methode getCurrentDuties() lädt die aktuellen diensthabenden Sanitäter/-innen herunter und fügt diese der angezeigten Liste hinzu. Dazu werden zunächst die Sanitäter/-innen durch eine Request, in der die Nutze-

rID und die SanitätsdienstID angegeben sind, heruntergeladen. Danach werden alle Einträge in dem JSON-Array durchgegangen und als DienstModel der Liste hinzugefügt. Dabei ist das Attribut isUser dafür da, dass der/die Nutzer/-in der App hervorgehoben wird, wenn er/sie selbst im Dienst ist. Danach wird noch überprüft, ob der/die Nutzer/-in die Berechtigungen hat den Dienst einzusehen und zu Vertreten / sich Auszutragen, sowie ob die auszuführende Aktion für den Button das Austragen oder das Vertreten ist, je nachdem ob die Person selbst gerade im Dienst ist oder nicht.

4.1.5 News & Notfallnummern

Um dem Sanitätsdienst eine gute Kommunikation zu ermöglichen, wurde zusätzlich ein News-Feature implementiert. Hier ist zurzeit nur das anzeigen von den News, die auf dem Server gespeichert sind möglich und nicht das Erstellen von neuen News. Die News werden genutzt um die interne Kommunikation des Sanitätsdienstes zu vereinfachen.

Um der alarmierenden Person zusätzliche Sicherheit zugeben ist außerdem ein View implementiert worden, auf dem die wichtigsten Notfallnummern aufgelistet sind, damit diese die entsprechende Nummer im Notfall durch anklicken der Nummer anrufen kann und sich nicht unbedingt an diese erinnern muss. Es wird hier nach dem Klicken ein AlertView angezeigt, bei dem man dann entscheiden kann ob man tatsächlich die angeklickte Notfallnummer wählen möchte, falls man sich verklickt hat.

4.1.6 Berechtigungen & Fehler

Um der Sanitätsdienst-Leitung möglichst viel Kontrolle zu geben, was von den Sanitäter/-innen gesehen und getan werden kann gibt es einige Berechtigungen, die auf dem Server gespeichert sind. Die einfachste und zugleich schwerwiegendste Berechtigung ist das Sperren von einem Account. Diese Berechtigung ist dafür gedacht, dass, sollte ein/e Sanitäter/-in oder ein/-e

Alarmierende/-r die App fälschlicherweise Nutzen, oder ähnliches kann dieser gesperrt werden.

Es gibt jedoch auch weniger drastische Mittel, mit denen die Nutzer/-innen eingeschränkt werden können, in dem was sie tun. Zum einen kann die Berechtigung entzogen werden mit hohen Prioritäten zu alarmieren(z.B. wenn die Person immer mit Priorität 5 alarmiert obwohl dies nicht notwendig wäre), oder sollte die Alarmierung so stark ausgenutzt werden, dass die Sanitäter/-innen unnötiger Weise alarmiert werden, kann die Berechtigung entzogen werden zu Alarmieren.

Außer bei der Alarmierung gibt es auch Berechtigungen bei dem Vertretungssystem. Hier gibt es die Möglichkeit den Sanitäter/-innen die Berechtigung zu entziehen zu vertreten oder sich aus dem Dienst auszutragen. Ein Anwendungsfall hierfür wäre zum Beispiel das sich ständige Austragen aus dem Dienst aus keinem Grund, oder dem Vertreten anderer Sanitäter/-innen ohne Absprache.

Die letzte Möglichkeit Nutzer/-innen einzuschränken ist, die Berechtigung zu entziehen, die eigenen empfangenen Alarme einzusehen. Dies könnte zum Beispiel aus dem Grund passieren, dass Alarmdaten an unbefugte Personen weitergegeben wurden und dies nun verhindert werden soll. Die Berechtigungen werden beim Server angefragt und dann je nach View umgesetzt. Der JSON-Array, der vom Server geladen wird, kann dann Folgendermaßen aussehen: [101, 102, 201, 202, 301, 302, 303, 304, 707, 708].

Um dem Nutzer aufzuzeigen, wenn ein Fehler auftritt, damit dieser z.B. nicht vergeblich auf eine/-n Sanitäter/-in wartet, werden sobald ein Fehler, z.B. bei einer http-Request, auftritt ein AlertView angezeigt, sodass der/die Nutzer/-in darauf reagieren kann.

4.2 Kommunikation mit dem Server

Um alle Funktionen anbieten zu können müssen einige Funktionen auf einen Server ausgelagert werden. Unter anderem werden hier die Sanitätsdienste

mit ihren Sanitäter/-innen und Alarmierenden verwaltet. Im Folgenden werde ich nun zu erst die API-Nutzung darlegen und im Anschluss Firebase-Messaging erklären, sowie auf die Implementierung von Firebase-Messaging eingehen.

4.2.1 API-Nutzung

Die API funktioniert so, dass ich zunächst eine Request, also eine https-Anfrage, an den Server sende. Hierbei spezifiziere ich zunächst den Pfad (/path) und danach werden Attribute in der query (?attribut1=wert1&attribut2=wert2) angegeben. Diese bestehen meistens aus der Sanitätsdienst-ID, der Nutzer-ID, der Nutzer-Rolle und weiteren spezifischen Attributen je nach Daten, die abgefragt werden sollen.

Der Aufbau einer solchen Anfrage würde dann wie folgt aussehen: Nach dem Empfang der Daten vom Server wird die Antwort des Servers zunächst in einem JSON-Array gespeichert um dann weiter verarbeitet zu werden. Dieser ist dann je nach Art der benötigten Daten 1-3 Dimensional. Zum Beispiel ist der JSON-Array für die Berechtigungen 1 Dimensional, der JSON-Array für die Sanitäter/-innen, die aktuell Dienst haben 2 Dimensional usw.

Durch die API ist die Programmierung deutlich einfacher und die App kleiner, da ich so alle Daten, die für die Nutzer/-innen nicht direkt relevant sind auf dem Server abspeichern kann, beziehungsweise, diese auch abfragen oder ändern kann, wodurch auf dem Handy selbst weniger Speicherplatz benötigt wird, da hier weniger Daten lokal gespeichert werden und weniger Systemressourcen benötigt werden, da mehrere Prozesse auch durch den Server erledigt werden.

4.2.2 Nutzung von Firebase-Messaging

Firebase-Messaging wird genutzt um die Alarme die, ausgelöst und vom Server verarbeitet werden, vom Server an die Smartphones der Sanitäter/-innen zu schicken. Hierzu habe ich mich entschieden, da Firebase-Messaging

mit den beiden Plattformen, iOS und Android, für welche auch meine App erhältlich ist, kompatibel ist und ich daher ohne Probleme Push-Notifications an die Smartphones der Sanitäter/-innen geschickt werden können. Da der Server, dies auch unterstützt und dafür die Möglichkeit bereitstellt, war dies schnell umgesetzt.

Firebase-Messaging ist ein Cloud-Messaging Service von Google. Dieser Service ist in der Lage Push-Notifications an Clients zu schicken. Dieses erfolgt über einen sogenannten FCM(Firebase-Cloud-Messaging)-Token, welcher einem spezifischen Gerät bei der Installation der App zugewiesen wird. Jeder FCM-Token ist einzigartig und wird von Zeit zu Zeit auf jedem Gerät aktualisiert. [12] Das Aktualisieren des Tokens kann durch mehrere Ereignisse ausgelöst werden. Zum einen dies dadurch ausgelöst werden, wenn die App auf einem anderen Gerät wieder hergestellt wird, oder der Nutzer die App deinstalliert bzw. diese reinstalliert, oder wenn der Nutzer die App-Daten löscht.

Firebase-Nachrichten sind grundlegend immer gleich aufgebaut. Grundlegend sind Firebase Nachrichten JSON-Arrays. Diese haben immer einen message-Teil, dieser kann dann noch weiter aufgedröselt werden. Ein wichtiger Teil, der in fast allen Nachrichten enthalten ist, ist der notification-Teil. In diesem wird dann der Notification-title und Notification-body angegeben, welche in der Notification angezeigt wird. Außerdem gibt es den data-Teil, welcher für die Daten Übermittlung zwischen Gerät und Server wichtig ist, da dieser selbst gestaltet werden kann. Durch den Server ist hier vorgegeben, dass bei jedem Alarm eine AlarmId, ein Alarm-Ort, eine Alarm-Beschreibung, eine Alarm-Priorität, die Zeit der Alarmauslösung, sowie das Datum von dem Alarm in dem Data-Teil des JSON-Arrays mitgegeben werden.

Firebase-Messaging stellt jedoch auch, wie sich im Laufe der Arbeit auch noch herausstellen wird eine Herausforderung dar, da der Programm-Code, welcher die Notification im Hintergrundmodus der App (also, wenn die App

gerade nicht geöffnet oder nur im Hintergrund geöffnet ist) empfängt und verarbeitet in einer anderen Isolate [2] ist, als der Code der den Vordergrund-Teil der App verarbeitet, wodurch unter anderem nicht auf die gleichen Daten zugegriffen werden kann, etc.

Um Firebase-Messaging zu nutzen muss zunächst auf der Firebase-Seite ein Projekt konfiguriert werden, in welchem festgelegt wird, welche Apps dem Projekt angehören und die Nachrichten vom Server erhalten. Danach müssen die Apps, die in diesem Projekt genutzt werden sollen konfiguriert werden. Um eine Android-App zu Konfigurieren muss der Android-Paketname und ein Name zur Identifizierung der App angegeben werden. Im Anschluss muss dann eine Konfigurationsdatei (google-services.json) heruntergeladen werden und dem Flutter-Projekt hinzugefügt werden (siehe Abbildung).

Das gleiche muss auch für iOS gemacht werden nur, dass der Ort und die Art der Konfigurationsdatei variiert (siehe Abbildung). Bei iOS ist es somit eine Info.plist-Datei anstatt der google-services.json-Datei.

Danach muss noch jeweils Plattform spezifischer Code eingefügt werden, worauf ich jetzt jedoch nicht weiter eingehen werde, da dieser von Firebase selbst vorgegeben wird und nur an der richtigen Stelle eingefügt werden muss. Um jetzt jedoch Firebase-Messaging nutzen zu können muss zunächst im Code die Firebase-App initialisiert werden und danach ein Background-Prozess gestartet werden, damit die Alarme auch wenn die App geschlossen ist empfangen werden können.

Um diesen Hintergrundprozess zu starten muss eine Methode erstellt werden, die die empfangenen Nachrichten verarbeitet. Dafür muss in der Methode FirebaseMessaging.onBackgroundMessage() eine globale Methode mitgegeben werden. Außerdem muss auf die Methode FirebaseMessaging.onMessage gehorcht werden. Dies geschieht durch den Aufruf von „.listen“ auf die Methode FirebaseMessaging.onMessage. In der Metho-

[2] Eine Isolate lässt sich mit Threads anderer Programmiersprachen und Betriebssysteme vergleichen

de listen() muss dann erneut die Nachricht, welche vom Server gesendet wurde verarbeitet werden. Außerdem muss auf die Methode Firebase.instance.onTokenRefresh gehört werden, um die Verarbeitung zu regeln, wenn der App ein neuer Token zugewiesen wird. Als letzte Methode muss dann noch auf die Methode FirebaseMessaging.onMessageOpenedApp() gehört werden, sodass verarbeitet werden kann wenn die App über das Klicken auf die Benachrichtigung geöffnet wird.

In diesem Projekt wird, wenn sich der Token abändert, der Token an den Server übermittelt, damit der/die Nutzer/-in jeder Zeit alarmiert werden kann, da sollte der Token der App unterschiedlich sein zu dem, der auf dem Server gespeichert ist, kann der/die Nutzer/-in nicht über die Firebase-Nachrichten erreicht werden.

In der listen-Methode von FirebaseMessaging.onMessageOpenedApp() wird zurzeit als einziges der Alarmsound gestoppt, ich habe mich dazu entschieden dies hier einzubauen, da die Sanitäter/-innen meist ihr Handy zunächst entsperren und im Anschluss dann auf die Benachrichtigung gucken, da die Sanitäter/-innen dann meist auf die Benachrichtigung klicken, da sie so die App öffnen können, hielt ich es am sinnvollsten das Stoppen des Sounds hier einzubauen, zusätzlich zu der Möglichkeit den Sound durch das Rückmelden in der App auszuschalten.

In der Methode onBackgroundMessage müssen deutlich mehr Daten verarbeitet werden. Zu aller Erst wird überprüft ob die Nachricht das das Attribut„Backpack“ mit dem Wert „true“enthält um zu überprüfen ob dies eine Benachrichtigung darüber ist, dass jemand anderes das Material holt oder ob die Firebase-Nachricht ein Alarm ist. Danach müssen die Alarmdaten, welche durch die Firebase-Nachricht empfangen wurden in die Datenbank geschrieben werden, um diese auch später noch anzeigen zu lassen. Zeitgleich wird außerdem der Alarmsound gestartet um den/die Sanitäter/-in über den Alarm zu informieren.

Zuletzt wird dann eine Request an den Server gesendet, in welcher darüber

informiert wird, dass der Alarm von der Nutzer/-in empfangen wurde. Dazu wird dem Server die NutzerID, die SanitätsdienstID und die Rückmeldung als int geschickt, in diesem Fall wird 0 mitgeschickt, da dies die Information ist, dass der Alarm empfangen wurde.

4.3 Speicherung der Daten

4.3.1 Umsetzung

Es gibt mehrere Möglichkeiten auf mobilen Endgeräten App-spezifische Daten zu speichern. Zum einen gibt es die so genannten SharedPreferences bzw. NSUserDefaults [3] dies sind einfache Schlüssel, mit denen ein Wert verknüpft wird. Eine weitere Möglichkeit ist eine lokale Datei, in welche alle wichtigen Daten geschrieben werden, oder als letzte, dritte Möglichkeit gibt es die Datenbank.

Im Fall der App DMergency habe ich zunächst versucht die Platform-Nativen Speichermethoden, also die SharedPreferences bzw. die NSUserDefaults, zu nutzen. Dies habe ich gemacht, da ich in der App selbst eigentlich kaum Daten speichern muss, was durch die später folgenden Skizzierung des Aufbaus der letztendlich genutzten Datenbank deutlich wird. Die SharedPreferences und NSUserDefaults sind generell sehr einfach gehalten, da die Daten über einen eindeutigen String identifiziert werden. Das heißt also es gibt einen SSchlüssel", der zu einem Wert zugeordnet wird. Die SharedPreferences bzw. NSUserDefaults können als Datentypen dann entweder einen String, einen int oder einen boolean zugewiesen bekommen. Wie bereits erwähnt habe ich zunächst diese Methode verwendet, da sie zum einen einfach ist, zum anderen aber auch kaum Daten gespeichert werden müssen.

Diese Methode musste ich jedoch schnell wieder verwerfen, da Flutter mit sogenannten Isolates arbeitet. Diese sind etwas ähnliches wie Threads, was bewirken soll, dass mehrere Funktionen gleichzeitig laufen können. Jedoch

[3] SharedPreferences (Android), bzw. NSUserDefaults(iOS) ist plattformspezifischer Langzeitspeicher für einfache Daten (String, Integer)

haben die SharedPreferences bzw. die NSUserDefaults dann pro Isolate eine eigene Instanz, was bedeutet, dass die Daten, die in der einen Isolate gespeichert wurden nicht von der anderen Isolate aus bearbeitet werden kann oder ähnliches. Dies ist ein Problem, da die Alarme, durch den Cloud-Messaging-Dienst Firebase-Messaging empfangen werden, dieser arbeitet dauerhaft in einer anderen Isolate als die Haupt-Isolate der App, da diese dauerhaft im Hintergrund laufen muss um die Alarme empfangen zu können. Da ich aber auf die Daten, die durch Firebase-Messaging gesendet werden angewiesen bin um diese in der App anzeigen zu können muss eine andere Lösung gefunden werden, um die Daten zu speichern, da ich keinen direkten Zugriff auf die Instanz von Firebase-Messaging habe.

Dadurch habe ich überlegt, dass ich die Daten dann in eine lokale Datei schreibe, um dann auf diese von jeder Instanz aus zugreifen zu können. Da dies jedoch eher Umständlich ist habe ich dann nach weiteren Möglichkeiten geguckt und bin zu dem Entschluss gekommen, dass die beste Möglichkeit, die Daten zu speichern, ist, diese in einer lokalen Datenbank-Datei zu speichern und die Daten dann von den jeweiligen Isolates abzuändern oder aufzurufen.

Dies ist letztendlich auch die Methode, die ich am Ende implementiert habe.

4.3.2 Aufbau der Datenbank

Im folgenden Abschnitt werde ich jetzt skizzieren, welche Daten in speichere, warum ich diese speichere und wie ich darauffolgend die Datenbank aufgebaut habe. Um die Funktion der App gewährleisten zu können müssen Daten des Nutzers gespeichert werden. Hierbei wird natürlich auf die Datenschutzbestimmungen geachtet. Diese besagen, dass alle erhobenen Daten nur für ihren angegebenen Zweck genutzt werden dürfen (Zweckbindung) und nur so viele Daten erhoben werden sollen wie benötigt werden(Datenminimierung)[14]. In meinem Fall speichere ich zunächst einmal den Namen, bzw. den Nutzernamen des Nutzers / der Nutzerin, um der Sanitätsdienst-Administration

zu ermöglichen die Nutzer/-innen zu identifizieren. Zudem wird die E-Mail des/der Nutzer/-in gespeichert, um die Nutzer/-innen eindeutig im System identifizieren zu können, daher: Die E-Mail-Adresse ist im gesamten Sanitätsdienst einzigartig. Außerdem wird das Geschlecht und die Qualifikation der Sanitäter/-innen gespeichert um den Administrator/-innen eine gute Dienstplanung zu ermöglichen. Außerdem wird zu jedem/-r Nutzer/-in die zugehörige Rolle gespeichert, also ob sie Sanitäter/-in oder Alarmierende/-r sind.

Es werden jedoch nicht nur die Login-Daten gespeichert, sondern auch die Daten der Alarme. Auf den Smartphones der Sanitäter/-innen werden jedoch nur die Daten gespeichert, die für den/die Sanitäter/-in relevant sind. Dies ist zum einen die Alarm-ID, welche zur eindeutigen Identifikation des Alarms benötigt wird, zum anderen werden aber auch der Alarmierungszeitpunkt(Datum & Zeit), die Beschreibung, der Ort und die Priorität gespeichert, um dem/der Sanitäter/-in möglichst viele Informationen über die Alarmierung zu geben. Der Alarmierungszeitpunkt ist zum Beispiel wichtig, wenn der Alarm verzögert kommen sollte, sodass der/die Sanitäter/-in weiß, dass eventuell Eile geboten ist, da schon mehr Zeit vergangen ist als eigentlich sollte. Als letztes wird für jeden Alarm die Rückmeldung des/der jeweiligen Sanitäter/-in die eigene Rückmeldung gespeichert, also ob der Alarm empfangen, Bestätigt oder Abgelehnt wurde.

Die Datenbank sieht dann wie folgt aus:

Die Tabelle Alarme hat wie in der Abbildung zu sehen eine Alarm-ID, welche ein Integer und zugleich der Primärschlüssel ist, da die Alarme mit ihrer Alarm-ID eindeutig identifiziert wird und somit einzigartig pro Sanitätsdienst ist. Die Beschreibung, der Ort, die Priorität, die Zeit und das Datum des Alarms werden als TEXT(String) abgespeichert, da diese theoretisch jede beliebige Zeichenkette enthalten können (sollen).

Die Tabelle Login hat eine Nutzer-ID, welche ein Integer und der

Primärschlüssel ist, da die Nutzer-ID den Login eindeutig kennzeichnet und über diese der/die Nutzer/-in eindeutig identifiziert wird. Außerdem gibt es das Attribut role, welches die Rolle des Nutzenden beschreibt. Hier nutze ich den Datentypen Integer um zwischen Sanitäter/-innen (role = 0) und Alarmierenden (role = 1) zu unterscheiden. Ich habe keinen boolean gewählt, da ich die Möglichkeit offenlassen möchte auch die Unterscheidung zum Administrator zu ermöglichen, da wie ich später noch ausführen werde noch einige Funktionen implementiert werden sollen, die bisher nicht implementiert sind. Des weiteren hat die Tabelle das Attribut „loggedin “, vom Datentyp TEXT, mit welchem ich überprüfe ob der Nutzer angemeldet ist oder nicht. Hierzu sollte eigentlich der Datentyp boolean verwendet werden, jedoch ist dieser von dem sqlite-package nicht supported[13]. Zudem gibt es eine sanID als Integer-Attribut, welches den Sanitätsdienst beschreibt, dem der/die Sanitäter/-in angehört. Des weiteren werden der Vorname, der Nachname die Qualifikation und das Geschlecht der Nutzerin / des Nutzers als TEXT gespeichert. Das letzte Attribut, welches gespeichert ist, ist volume, welches vom Datentyp REAL ist. In diesem speichere ich einen double, welcher die in den Einstellungen festgelegte Lautstärke für Alarme speichert.

Dies kann in der unten aufgeführten Abbildung nochmal entnommen werden.

Wie bereits durch meine Ausführungen deutlich geworden sein sollte, wurde bei jeder erhobenen Information darauf geachtet, dass nur die Daten erhoben werden, die für die Funktion der App von Nöten sind, wodurch die DSGVO in der App eingehalten wird. Diese gibt vor, dass alle erhobenen Daten nur für den angegebenen Zweck genutzt werden dürfen und nur Daten erhoben werden sollten, die unbedingt genutzt werden müssen[14].

5 Schultests

Um die Funktionalität der App sicherzustellen wird die App zur Zeit an mehreren Schulen getestet. Dazu wird für die Schulen ein Sanitätsdienst erstellt, über welchen diese dann alle Funktionen testen können. Sollte dann ein Fehler auftreten können die Nutzer/-innen diesen über ein Fehler-Formular melden. Jedoch kann man nicht nur Fehler melden sondern auch generell Feedback, über ein dediziertes Fehler-Formular, abgeben. Dies dient der allgemeinen Verbesserung der App in jeglichen Belangen, seien es fehlende Funktionen, Design-Abänderungen oder nur Lob. Derzeit testen bereits 5 Schulen die App, wobei generell positives Feedback zurückkam. Jedoch trat natürlich auch schon der ein oder andere Fehler auf, der dann jedoch durch schnelle Kommunikation mit den jeweiligen Ansprechpartner/-innen der Schulen behoben werden konnte. Zum Beispiel wurde bereits die Wahrscheinlichkeit erhöht, in der ein Alarm ankommt. Hier gab es vor allem am Anfang größere Probleme, da immer wieder Sonderfälle aufgetreten sind, durch die Fehler aufgetreten sind. Dadurch, dass in der App dann Fehler aufgetreten sind konnte der Alarm nicht abgespielt und angezeigt werden. Dies konnte dann durch die Rückmeldungen der Schulen behoben werden. Andere Fehler waren beispielsweise auch Grafikfehler, bei denen dann gewisse Grafikobjekte nicht korrekt angezeigt wurden oder abgeschnitten waren. Dies konnte dann auch schnell durch die Rückmeldungen behoben werden. Des weiteren wurde dann auch ein Fehler beim Anmelden gemeldet, durch welchen man sich weiterhin registrieren konnte, jedoch eine Anmeldung mit einem existenten Account nicht möglich war. Durch mehrere schnelle Tests war dann schnell klar, dass der Fehler durch die falsche Verarbeitung der Serverantwort aufgetreten ist, sodass auch dieser schnell behoben werden konnte. Diese schnelle Fehlerbehebung war zum einen durch das bereits erwähnte Fehlerformular möglich, jedoch auch durch einen weiteren Service von Firebase. Durch Firebase-Crashlytics ist es möglich, dass Fehler, die in der App auftreten auf einer Web-Oberfläche angezeigt werden. Durch die Im-

plementation von Firebase-Crashlytics ist durch die Fehler-Meldungen eine Lösungsfindung deutlich einfacher, da hier der Fehler zum einen angezeigt wird und zum anderen angezeigt wird, an welcher Stelle dieser ist.

6 Fazit

In den folgenden zwei Abschnitten werde ich jetzt ein Fazit ziehen, in dem ich zunächst aufzeige, was erreicht wurde und im Anschluss einen Ausblick gebe, was noch ansteht, beziehungsweise, welche Funktionen vielleicht noch hinzugefügt werden sollen o.ä.

6.1 Was wurde erreicht

6.2 Wie geht es weiter

7 Abbildungsverzeichnis

Abbildungsverzeichnis

8 Literaturverzeichnis

Literatur

[1] https://www.drk.de/hilfe-in-deutschland/erste-hilfe/rettungskette/rettungskette-uebersicht/

[2] https://www.android.com/, Offizielle Android-Seite

[3] https://www.apple.com/de/ios/ios-15/, Offizielle iOS-Seite

[4] https://de.statista.com/statistik/daten/studie/256790/umfrage/marktanteile-von-android-und-ios-am-smartphone-absatz-in-deutschland/#: :text=Marktanteile%20von%20Android%20und%20iOS,2021&text=Im%203%2DMonatszeitraum%20Oktober%20bis,iPhone%20betrug%2030%2C9%20Prozent.

[5] https://docs.microsoft.com/en-us/xamarin/xamarin-forms/,Xamarin Forms Dokumentation

[6] https://dart.dev/, Dart Dokumentation

[7] https://www.flutter.dev, Dokumentation des Flutter-Frameworks

[8] https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html, Dokumentation der Klasse Stateless-Widget

[9] https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html, Dokumentation der Klasse Stateful-Widget

[10] , https://api.flutter.dev/flutter/widgets/State-class.html, Dokumentation der Klasse State

[11] https://dart.dev/codelabs/async-await, Dokumentation von Asynchronität

[12] https://firebase.google.com/docs/cloud-messaging/manage-tokens, Aktualisierung von FCM-Token, Firebase-Dokumentation

[13] https://pub.dev/packages/sqflite#: :text=Supported%20SQLite %20types%20%23, sqlite - Unterstützte Datentypen

[14] https://www.datenschutz-grundverordnung.eu/grundverordnung/ art-5-ds-gvo/, DSGVO-Artikel 5

9 Anhang

Hiermit erkläre ich, Mattis Rinke, dass ich diese Ausarbeitung ohne fremde Hilfe angefertigt habe und nur die im Literaturverzeichnis aufgeführten Quellen und Hilfsmittel genutzt wurden.

....., den .....

(Ort) (Datum) (Unterschrift)