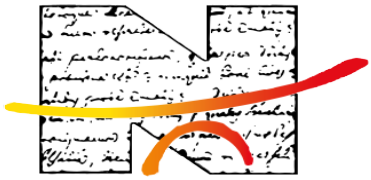
Gymnasium Nepomucenum Coesfeld
Besondere Lernleistung im Abitur 2022



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

In der heutigen Welt wird der Drang nach Digitalisierung immer größer, wie auch bei Sanitätsdiensten. Die habe ich selbst durch meine Tätigkeit im Schulsanitätsdienst erfahren. 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 anschließend dann die Sanitäter/-innen alarmiert.
Um hier Verbesserung zu schaffen, habe ich mich dann nach Apps für eine einfache Alarmierung umgeguckt. Dabei habe ich dann mehrere Apps gefunden unter anderem die Apps SSanialarmünd "Divera 24/7", beide haben die grundsätzlichen Voraussetzungen um für den Schulsanitätsdienst genutzt zu werden: Beide Apps können Alarmieren und einen Dienstplan erstellen. Jedoch haben beide auch ihre Einschränkungen, wodurch diese nicht komplett für den Schulsanitätsdienst geeignet sind. Zum Beispiel ist die Alarmierung von Sanialarm nicht zuverlässig und die Erstellung des Dienstplans muss bei Divera für jeden Tag einzeln erfolgen. Aufgrund dieser Einschränkungen habe ich mich zusammen mit einem Freund darangesetzt diese Problem zu lösen. Durch meine Vorkenntnisse im Fach Informatik bin ich dann schnell auf die Idee gekommen die Alarmierung per App zu gestalten. Ich habe mich dann, als ich bereits angefangen hatte die App zu programmieren von der Möglichkeit erfahren eine besondere Lernleistung in das Abitur einfließen zu lassen, dazu entschieden dies zu tun.
In dieser Ausarbeitung gehe ich darauf ein, wie die App entstanden ist, 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 warum können soll und beschreibe im Anschluss, wie ich mich von der nativen Entwicklung zur Nutzung des Cross-Platform-Frameworks Flutter entschieden habe. Danach erkläre 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 erkläre, was in der Ausarbeitung bereits geschafft wurde und wie ich mit der App weiterhin verfahre.

2 Das Projekt

2.1 Zielsetzung

Die App soll das Alarmieren und Verwalten von Sanitätsdiensten vereinfachen. Um dies zu verwirklichen müssen mehrere Funktionen implementiert werden. Zum einen muss es eine Funktion zum Alarmieren geben, welche zur Vergewisserung für die Alarmierende Person auch ein Feedback anzeigen sollte, zum Anderen sollte es dann logischer Weise auch eine Funktion zum Empfangen des Alarms geben. Diese beiden Funktionen sollten so implementiert werden, dass eine alarmierende Person so wenig Aufwand wie möglich beim Alarmieren hat, die Sanitäter/-innen jedoch so viele Informationen wie möglich bekommen. Damit die Sanitäter/-innen sich zusätzlich darüber verständigen können, wer das Einsatzmaterial holt, sollte dies auch mit in diese Funktionen aufgenommen werden. Um zu definieren, wer einen Alarm wann erhält soll außerdem ein Dienstplan System programmiert werden, damit nicht immer alle Sanitäter/-innen alarmiert werden. Da es aber auch immer Notfallsituationen gibt, in denen einzelne Sanitäter/-innen nicht erreichbar sind, erfordert dies, dass eine Funktion eingebaut wird, durch die einzelne Sanitäter/-innen von anderen Sanitäter/-innen vertreten werden können oder sich die Sanitäter/-innen, sollten sie keine Vertretung finden o.ä., austragen können. Um die Hilfskette[1] möglichst kurz zu halten sollen in der App wichtige Notfallnummern hinterlegt werden, welche dann durch einen schnellen Klick auch wählbar sind. Ein weiterer wichtiger Bestandteil zur Verwaltung des Sanitätsdiensts ist die Kommunikation zwischen der Leitung und den Mitgliedern des Sanitätsdiensts. Um diese Kommunikation

sicherzustellen soll eine News-Funktion programmiert werden, in welcher die Mitglieder Neuigkeiten von der Leitung einsehen können. Die Leitung muss dann natürlich News schreiben 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 funktioniert 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. Als ich mit dem Projekt angefangen habe, habe ich zunächst den mir am sinnvollsten erscheinenden Weg genommen. Die native Entwicklung. Dadurch, dass die App für unterschiedliche Betriebssysteme erhältlich sein soll, muss dies in dem Fall dann zweimal geschehen. Einmal für das Betriebssystem Android[2], von Google, und für das Betriebssystem iOS[3] von Apple. Es gibt aber auch die Möglichkeit der Cross-Platform-Programmierung, bei der für beide Betriebssysteme gleichzeitig programmiert wird. Ich habe mich zunächst für die native Programmierung entschieden, jedoch habe ich mich im Entwicklungsprozess von der nativen Entwicklung zur Cross-Platform-Programmierung mit dem Framework Xamarin-Forms bewegt um schließlich das Framework Flutter zu verwenden. Was die Vor- und Nachteile sind und warum ich mich letztendlich für die Cross-Platform-Programmierung mit Flutter entschieden habe erkläre ich in den nächsten drei Abschnitten.

3.1 Version 1 - Native Entwicklung

Zunächst habe ich mit der nativen Entwicklung von Android begonnen, da ich die eine garantierte Kompatibilität mit dem gewünschten Betriebssystem habe. Dies stellte sich als einfach heraus, da ich hier in der Programmiersprache Java schreiben muss, die ich bereits aus dem Informatik-Unterricht kannte. Hier hatte ich dann nach einiger Zeit eine vor**läufig fertige App programmiert, in welcher man einen Alarm auslösen und empfangen konnte. Da der Markt zwischen Apple und Google in Sachen Handy-Betriebssysteme bei ca. 70% zu 30% liegt[4], habe ich schnell gemerkt, dass ich die App auch für iOS entwickeln muss. Um alle Funktionen immer auf jeder Platform verfügbar zumachen muss bei dieser Entwicklungsmethode jede Funktion zweimal programmiert werden. Dies ist für mich als Einzelperson nicht machbar, weshalb ich mich neu orientieren musste. Ich habe mich weiter informiert und habe dann die Methode des Cross-Platform-Programmings gefunden, auf welche ich in den nächsten zwei Abschnitten eingehen werde.

3.2 Version 2 - Xamarin Forms

Xamarin Forms ist ein Framework der .NET-Platform 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]. 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, zum anderen musste ich jedoch auch herausfinden, 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 gehe ich im Laufe der Ausarbeitung noch ein). 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, sowie nach eigener Recherche. Flutter nutzt eine einfache Programmiersprache, die Java ähnelt, weshalb mir der Einstieg in die Programmiersprache Dart[6] nicht schwer gefallen ist. Flutter ist wie Xamarin-Forms 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 wird in Flutter durch so genannte Widgets, welche in einem Widget-Tree aufgenommen werden, dargestellt. Hierbei wird dann zwischen Stateless[8]- und Stateful[9]-Widgets unterschieden. Der Unterschied hier ist, dass in Stateless-Widgets zwar Variablen, etc. abgeändert werden können, jedoch wird das Widget im UI nicht aktualisiert. Es können aber in dem Widget selbst andere Widgets, die selbst Stateful-Widgets sind, aktualisiert werden. Ein Stateful-Widget besteht aus zwei Klassen. Der Klasse, die um die Klasse Stateful-Widget erweitert wird und der Klasse, die um die Klasse State erweitert wird. In der Klasse mit dem Stateful Widget wird in der createState-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 dem der Widget-Tree dann erstellt wurde kann bei Bedarf durch die Methode setState das UI aktualisiert werden. Außerdem gibt es asynchrone Methoden und Futures[10]. Diese sind dazu da das Programm weiterhin Code ausführen zu lassen, während

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

dieses 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. Ich habe mich nach einiger Überlegung und Recherche dann letztendlich für Flutter als Framework entschieden, weil ich zum einen nur einen Code für die beiden Plattformen iOS und Android schreiben muss und zum anderen da das UI einfach programmierbar ist 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 normale 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)
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 sich 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. In den Betriebssystemen Android und iOS, gibt es zum einen Funktionen, die ohne jegliche weitere Berechtigung genutzt werden können. 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 Accountnamen festlegen. Dieser besteht jedoch nicht aus Vor- und Nachname sondern nur aus einem generalisierten Namen, da hier theoretisch auch Ac-

countnamen 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 Prioritäts-Auswahl 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 angezeit.

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 „Alarminfo “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 Rei-

henfolge 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.

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

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].

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.

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.[11] 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.

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 [2] 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 folgende Skizzierung des Aufbaus der letztendlich genutzten Datenbank deutlich wird. Die

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

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 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 Hauptisolate 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)[13]. 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 emp-

fangen, 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[12]. 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[13].

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.

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://dart.dev/codelabs/async-await, Dokumentation von Asynchronität

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

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

[13] 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)