St.-Anna-Schule Wuppertal
Q2 2021
Herr Gerd Heischkamp
Abgabe:

Programmierung einer React Native App, mit Firebase als Backend und dem Schwerpunkt Datensicherheit

Finn Malkus
finn@malkus.de
Q2 20/21

Gliederung

1. Vorwort

Dieses Dokument dient als Dokumentation der App „GERD“, welche ich im Rahmen meiner besonderen Lernleistung für mein Abitur 2021 programmiert habe. Das Projekt ist Open Source und daher auf der Plattform GitHub zu finden unter: https://github.com/FinnMal/Gerd. Zum Stand der Abgabe ist die App noch nicht im App- und PlayStore veröffentlicht. Für die Zukunft ist das aber geplant. Die hier gezeigten Screenshots zeigen das helle Design der App. Zusätzlich kann in den Systemeinstellungen das dunkle Design (Dark Mode) aktiviert werden. Der Dark Mode ist für iPhones ab IOS 13 verfügbar (vgl. HEISE ONLINE – TIPP: DARK MODE BEI WHATSAPP AM IPHONE AKTIVIEREN, 2020, URL).

2. Idee

Die Kommunikation von Vereinen und Organisationen zu den Mitgliedern verläuft meist digital per Mail und in einigen Fällen sogar noch analog per Post oder Flyer. Für Vereine ist es wichtig in Konakt zu den Mitgliedern zu bleiben. Sei es um aktuelle Veranstaltungen zu vermarkten oder die Noten für das nächste Chortreffen per Mail zu verschicken. Der Anwendungsbereich ist vielseitig. Die Kommunikation verläuft jedoch ebenso unterschiedlich. Der eine Verein verschickt wichtige Informationen per Newsletter, der andere lädt die Notentexte auf externen Servern hoch. Und im schlimmsten Fall kommen die Mitteilungen sogar noch beim falschen Empfänger an. Der Datenschutz bleibt dabei auch meist auf der Strecke. Diese Zenarien erleben viele Vereine und Organisationen täglich. Die Prozesse sind lästig für die Vereinsbetreiber, aber vorallem für die Vereinsmitglieder. Die gezwungen sind sich durch verschiedene Mails und Websites durchzuwühlen, nur um eine einzige Datei zu finden. Zudem gibt es auch keine wirklichen Alternativen. Die Kommunikation per WhatsApp fällt zum Beispiel aus Datenschutzgründen weg und andere Messenger-Apps sind für ältere Mitglieder meist zu kompliziert. Die Idee der GERD-App ist es daher ein einheitliches Tool zu bieten, welches all diese Anwendungsbereiche vereint und auf die Kommunikation für Vereine spezialisiert ist. Außerdem gibt es die Möglichkeit per Push-Notifications

Mitglieder zu informieren, Termine zu planen, Dateien zu verbreiten und im privaten verschüsseltem Chat mit einzelnen Mitgliedern zu kommunizieren. Das alles auf der Grundlage einer einfachen Bedienbarkeit und Anonymität.

3. Strukturierung

Die App besteht aus der IOS-/Android Anwendung basierend auf React Native, welche der Benutzer auf seinem Smartphone installiert, dem Firebase Backend, welches auf Google Servern in Iowa (USA) läuft (vgl. GOOGLE CLOUD – BUCKET-STANDORTE, 2020, URL), und dem OneSignal Backend, was die Übermittlung der Push-Notifications übernimmt.

Firebase ist ein kostenloses Tool von Google. Es basiert auf der Funktionalität der Google Cloud Server und bietet ein einfaches Backend für Web- und Mobileanwendungen. Für das Projekt wurde die Firebase-Echtzeitdatenbank verwendet, sowie das Node.js Backend „Firebase Functions“.

3.1 Firebase Datenbank

Die „Firebase Realtime Database“ basiert auf einer JSON-Struktur und kann sowohl per REST als auch direkt über JavaScript abgefragt werden. Für letzteres ist eine Erweiterung von Firebase notwendig. Die Anfragen auf die Datenbank setzten immer eine vorherige Authentifizierung voraus (Siehe Abschnitt 7.2). Mit Firebase ist die Registrierung per E-Mail, Facebook, Google oder anonym möglich. Für dieses Projekt wurde die anonyme Authentifizierung verwendet. Um, vor allem für ältere Benutzer, keine zusätzlichen Hürden während der Registrierung zu verursachen (Siehe Abschnitt 7.1). Der Großteil der Kommunikation zu den Firebase-Datenbank-Servern läuft über den „DatabaseConnector“ (Siehe Abschnitt 8.1).

In der Firebase Datenbank werden Vereinsinformationen, Benutzerdaten, und Informationen zu Chats gespeichert. Der Zugriff auf die verschiedenen Datensätze ist durch die „Firebase-Database-Rules“ beschränkt (Siehe Abschnitt 7.2).

Durch sogenannte „Database Listener“ kann innerhalb der App in „Echtzeit“ auf Änderungen in der Datenbank reagiert werden. Diese Funktion kommt zum Beispiel beim Ändern von Benutzerinformationen, Dateien oder Veranstaltungsinformationen zum Einsatz. Dies gewähleistet, dass bei allen Benutzern korrekte und aktuelle Inhalte angezeigt werden.

3.2 Firebase Functions

Die „Firebase Functions“ sind ein programmierbares Backend basierend auf Node.js, welches auf den Google Servern ausgeführt wird. Firebase Functions sind mit sogenannten Triggern ausgestattet, die die erstellten Algorithmen ausführen, sobald das, mit dem Trigger verknüpfte Event, eintritt. Auslöser für Trigger können das Hinzufügen, Entfernen oder Ändern von Einträgen in der Firebase Datenbank sein, bestimmte Intervalle und Uhrzeiten, oder gewöhnliche HTTP-Requests.

Dieser Trigger reagiert zum Beispiel auf neue Mitteilungen eines Clubs:

```
1. exports.sendMessageNotifications =
2. functions.database.ref('clubs/{club_id}/messages/{mes_id}').onCreate(async (snapshot, context) => {console.log('message added') });
```

Es wird eine neue Funktion mit dem Namen „sendMessageNofitications“ erzeugt, welche auf Änderungen innerhalb eines Club-Objekts reagiert. Teile eines Pfades lassen sich durch Variablen ersetzten, um auch auf Änderungen verschiedener Child-Objekte reagieren zu können. Auf die Pfad-Variablen kann in der Funktionsausführung zugegriffen werden. Nach der Angabe des Pfades folgt ein Ereignishandler. Dieser bestimmt auf welche Art von Änderung in der Datenbank reagiert werden soll. Hier kann zwischen „onWrite“ (Daten werden erstellt, aktualisiert oder gelöscht), „onCreate“ (Daten werden erstellt), „onUpdate“ (Daten werden aktualisiert) und „onDelete“ (Daten werden gelöscht) ausgewählt werden. Für dieses Projekt werden die Firebase Functions verwendet, um QR-Codes für Vereins-Invites zu generieren (Siehe Abschnitt 6.2), Push-Notifications, für zum Beispiel Chat-Nachrichten oder Veranstaltungen, zu versenden und wiederholende Veranstaltungen zu planen.

3.3 OneSignal

Für das Versenden von Push-Mitteilungen wird der Dienst von OneSignal verwendet. OneSignal ist ein Anbieter zum Versenden von Push-Notifications auf verschiedene Web- und App Plattformen. Für dieses Projekt wurde die von OneSignal bereitgestellte Erweiterung für React Native Apps verwendet. OneSignal kann mithilfe von HTTP-Requests von den Firebase Functions angesprochen werden. Der Empfang der Push-Notifications erfolgt außerhalb der eigentlichen React Native App und wird im Hintergrund ausgeführt, unabhängig davon, ob die App geöffnet ist oder nicht. Klickt der Benutzer auf eine Benachrichtigung, oder wird eine Benachrichtigung empfangen während die App geöffnet ist, kann darauf mithilfe von Callbacks durch die React Native Instanz reagiert werden.

Beispiel für das Erstellen eines OneSignal-Callbacks in App.js:

```
OneSignal.addEventListener("opened", this.onOpened);
```

Jedem Nutzer ist eine eigene OneSignal-ID zugewiesen, welche beim ersten Start der App von OneSignal generiert wird und anschließend in der Firebase Datenbank gespeichert wird (Siehe Abschnitt 6.1). Somit lassen sich auch Push-Notifications an einzelne Nutzer senden.

Die Benutzer können zudem durch Tags identifiziert werden. Die Tags werden gesetzt, sobald ein Benutzer einem Verein beitritt. Mithilfe der Tags können die Nutzer den verschieden Vereinen zugeordnet werden, um die Push-Notifications gezielt zu versenden.

Beispiel für das setzen eines OneSignal-Tags:

```
OneSignal.sendTag("NAME", "VALUE");
```

4. React Native

React Native ist ein Framework zum Entwickeln von mobilen Anwendungen für IOS, Android, und Webplattformen. Der Quellcode wird ausschließlich in JavaScript geschrieben und ist für alle Plattformen gleich. Das Framework basiert auf React, der JavaScript-Bibliothek von Facebook zum Erstellen von

Benutzeroberflächen. React Native rendert, basierend auf dem JavaScript-Quellcode, native Objekte. Dadurch ist für den Benutzer kein Unterschied zu gewöhnlichen Apps sichtbar. Das Framework wird unteranderem für die Apps von Facebook, Instagram, Skype, Tesla und Pinterest verwendet. Für die jeweilige Plattform werden von React Native Projekt-Ordner erstellt. Bei IOS ein Xcode Projekt und für Android ein Android Studio Projekt. Die Projekte beinhalten den Code zum ausführen des React Native Frameworks, sowie alle nötigen JavaScript-Packages (Siehe Abschnitt 4.2.2). Die erstellten Projekte beinhalten jedoch nicht den JavaScript Quellcode selbst, dieser wird beim Ausführen der App aus den js-Dateien gelesen und ausgeführt. Der JavaScript-Code wird von React Native also nicht in Nativen Code umgewandelt. Durch die Ausgabe der Projekte lassen sich auch Änderungen am Nativen Code vornehmen und APK bzw. IPA Dateien erstellen.

4.1 Struktur des Projektordners

Der Ordner eines React Native Projekts enthält die Quellcodes als JavaScript-Dateien (.js), Konfigurationsdateien (.json), und Ordner für die jeweiligen Plattformen, hier IOS („ios“) und Android („android“) (Siehe Abbildung 6). Die Ordner „app“, „classes“, „components“ und „screens“ werden nicht von React Native erstellt. Sie dienen zur Ordnung der JavaScript-Dateien.

Die Plattform-Ordner werden einmalig zu Beginn der Entwicklung mit folgendem Befehl erstellt:

```
react-native eject
```

4.2 Entwicklung mit React Native

4.2.1 Programmierung

Die Programmierung der eigentlichen App wird in den jeweiligen JavaScript-Dateien vorgenommen. Die Ausführung der App startet mit der Datei „App.js“. Hier wird der Navigator bestimmt. Im Navigator sind alle Screens der App eingetragen und bestimmt, wie die Navigation erfolgt. Beispielsweise durch rechts wischen oder links wischen. Im Navigator wird auch der Start-Screen bestimmt, also der Screen, der beim öffnen der App als erstes angezeigt wird.

Bei diesem Projekt ist der Start-Screen „ScreenHandler.js“ (Siehe Abschnitt 8.3). In App.js können zudem weitere Aufgaben erledigt werden, die zur Intialisierung der App wichtig sind. Beispielsweise wird die Verbindung zu Firebase und OneSignal hergestellt. Als nächstes werden die Elemente in ScreenHandler.js gerendert. Hierzu ist es wichitg, die Struktur eines Elements in React Native zu kennen. Diese beginnt zunächst mit dem Konstruktor, in dem die Attribute („props“) an das Objekt übergeben werden.

Der standart Konstruktor eines React-Elements sieht folgendermaßen aus:

```
1.  constructor(props) {
2.          super(props);
3.   }
```

Werden keine Anpassungen im Konstruktor vorgenommen, so kann dieser auch weggelassen werden. Nach dem Erstellen des Elements wird von React die „render“-Methode aufgerufen. Hier wird, ähnlich wie bei HTML, die Strukstur der Elemente angegeben. Ausschnitt der „render“-Methode vom Element ClubCard.js:

```
1.  render() {
2.    return (
3.      <Theme.TouchableOpacity
4.        style={{
5.          marginBottom: 20,
6.          borderRadius: 14,
7.          padding: 11,
8.          backgroundColor: "#" + this.props.club_color,
9.          shadowColor: "#" + this.props.club_color,
10.         flexDirection: 'row',
11.         alignItems: 'center',
12.         justifyContent: 'center'
13.       }}
14.       onPress={() => this.onPress()}>
15.       ...
16.     </Theme.TouchableOpacity>
17.   );
18. }
```

Deutlich wird, dass es sich bei der „render“-Methode um eine Mischung aus HTML, CSS und JavaScript handelt. Die Objekt-Attribute, wie zum Beispiel „style“ oder „onPress“ werden dem Objekt durch das „props“-Objekt mitgegeben. Möchte man beispielsweise die übergebene „onPress“-Funktion aufrufen, so benötigt man folgenden Code-Ausschnitt:

```
this.props.onPress()
```

Will man den programmierten Code auf einem Smartphone testen, so ist dies zum einen über die Plattform Expo möglich, aber auch über den sogenannten „Metro Bundler“ von Facebook. Expo bietet eine Vorauswahl an Packages an und kann innerhalb der Expo App selbst getestet werden, die eigene App muss somit nicht über Xcode oder Android Studio compiliert werden. Jedoch ist es auch nicht möglich zusätzliche Packages zu installieren, die zum Beispiel für die Integration von OneSignal nötig sind. Daher wurde für dieses Projekt der Metro Bundler verwendet. Der Metro Bundler basiert auf Node.js und startet sich automatisch, sobald man die App in Xcode auf dem iPhone oder einem Emulator startet. Öffnet man nun die App, wird der Quellcode über den Metro Bundler heruntergeladen, wie in Abbildung 1 sichtbar. Die Übertragung läuft über das lokale Netzwerk.

Abbildung 1: Download über den Metro Bundler

Der Metro Bundler reagiert zudem auf Änderungen an dem lokalen Projekt und läd automatisch die neue Version auf das Testgerät. Somit muss nicht bei jeder Änderung am Quellcode die App neu compiliert und auf dem iPhone installiert werden. Zudem stürtzt die App nicht ab, sobald ein Bug auftriggt, wie es zum Beispiel bei Xcode der Fall ist. Fehlermeldungen werden direkt in der App angezeigt. Ein Beispiel für eine Fehlermeldung innerhalb der App ist in Abbildung 7 zu finden.

Nachdem die App erfolgreich getestet und fehlerfrei ist, kann in Xcode die Release-Version erstellt werden. Diese basiert dann nicht mehr auf dem Metro

Bundler und kann als eigenständige App im AppStore angeboten werden. Fehlermeldungen werden dann auch nicht mehr angezeigt.

4.2.2. Installation eines Pakets

Wie auch in anderen Programmiersprachen üblich kann das eigene Projekt durch die Installation externer Pakete erweitert werden. Bei React Native ist hierfür der Node Package Manager (npm) vorgesehen. Pakete können wie üblich über folgenden Befehl installiert werden:

```
npm install (...)
```

Um das Paket nach der installation mit React Native verwenden zu können muss es mit dem jeweiligen nativen Projekt verknüpft werden. Beim IOS Projekt ist das mithilfe von „CocoaPods“ (pod) möglich. CocoaPods verknüpft die mit npm installierten Pakete mit dem Xcode Projekt. Die Abhängigkeiten, also welche Pakete mit dem Xcode Projekt verknüpft werden müssen, werden in der Datei „Podfile“ festgelegt, die sich im Ordner des IOS-Projekts befindet. In der Regel sind hier aber keine Änpassungen nötig, da die installierten npm-Pakete automatisch ausgelesen und verknüpft werden.

Folgender Befehl startet die Verknüpfung mit CocoaPods, dieser muss innerhalb des IOS-Projektordners ausgeführt werden:

```
pod install
```

Nach der Verknüpfung muss das Xcode-Projekt neu compiliert und auf dem Gerät installiert werden.

4.3. Vor-/Nachteile

Die plattformübergreifende Programmierung ist eines der Vorteile der React Native Plattform. Sie spart zum einen viel Arbeit, lässt die App aber auch, durch gleiches Design und Funktionen, auf beiden Plattformen einheitlicher wirken. Wird von IOS beziehungsweise Android eine neue Funktion für Apps im Betriebsystem veröffentlicht, so dauert es bei React Native jedoch einige Wochen bis Monate, bis es verwendet werden kann. Da man auf ein von der Community erstellte Erweiterung angewiesen ist. Man hat aber auch die Möglichkeit den nativen Code der App selbst zu erweitern und somit die

Funktionalität mit seinem React Native Projekt zu verknüpfen. Die Verwendung von React Native setzt Grundlagen vorraus, die die meisten Programmierer bereits mitbringen: HTML, CSS und JavaScript. Laut Satista war JavaScript im April 2021 auf Platz 3 der beliebtesten Programmiersprachen weltweit (vgl. STATISA – DIE BELIEBTESTEN PROGRAMMIERSPRACHEN (...) IM APRIL, 2021, URL). Durch Expo und dem Metro Bundler ist das Testen und die Fehlerbehebung auf dem eigenen Gerät sogar einfacher als mit den herkömmlichen Tools wie Xcode oder Android Studio. React Native profitiert zudem von einer sehr großen Community, das Projekt wird also stetig weiterentwickelt.

5. Funktionen und Screens in der App

Die App bietet umfangreiche Möglichkeiten zur Kommunikation mit Vereinsmitgliedern. Es ist zwischen den Funktionen für Vereinsbetreiber und Vereinsmitglieder zu unterscheiden. Der Account des Vereinsbetreibers ist im Grunde eine erweiterte Version eines normalen Accounts. Der Funktionsumfang wird also nur durch die Verwaltungsmöglichkeiten für die Vereine erweitert. Spezielle Vereinsaccounts, bei denen die Verwaltung des Vereins im Vordergrund steht, gibt es nicht. Durch das Menü (Siehe Abbildung 9, unten im Bild) kann zwischen den einzelnen Bildschirmen gewechselt werden. Alle Bildschirme die über das Menü zugänglich sind, werden über den ScreenHandler verwaltet, und je nach Verwendung ein und ausgeblendet (Siehe Abschnitt 8.3).

5.1 Startseite

Für alle Benutzer wird auf der Startseite die Übersicht der aktuellen Veranstaltungen und Mitteilungen angezeigt (Siehe Abbildung 9). Oben rechts können die Benutzer auf ihr Profilbild klicken und gelangen somit zu einer Übersicht über die letzten Push-Notifications. Diese Funktion ist zum jetzigen Stand jedoch noch nicht umgesetzt. Direkt unterhalb des Profilbildes werden die aktuellen Veranstaltungen der Vereine angezeigt. Hierbei handelt es sich um einen selbst entwickelten „Slider“. Nach vier Sekunden wird die angezeigte

Veranstaltung automatisch nach links rausgeschoben und die nächste Veranstaltung wird angezeigt. Durch wischen kann ebenfalls die nächste Veranstaltung angezeigt werden. Die automatische Bewegung wird damit ausgeschaltet. Als Hintergrund können Vereine eigene Bilder hochladen, oder der Server wählt zufällig eines der Standartmuster aus. Scrollt man nun weiter runter, werden die letzten Mitteilungen der Vereine angezeigt. Hat der Nutzer bereits auf eine Mitteilung geklickt, so wird diese vom Startbildschirm entfernt und in einer extra Übersicht einsortiert, die über de Button „Alle anzeigen“ aufgerufen werden kann.

5.2 Mitteilung

Nach dem Klicken auf eine Mitteilung öffnet sich der Mitteilungs-Screen (Siehe Abbildung 8). Der Benutzer kann nun den gesamten Text der Mitteilung lesen. Scrollt man weiter runter, so lassen sich verknüpfte Dateien runterladen, oder Veranstaltungen anzeigen. Oben wird der Autor der Mitteilung mit Name und Profilbild angezeigt, sofern dies vom Autor selbst beim Senden der Mitteilung so festgelegt wurde. Mitteilungen lassen sich auch anonym verfassen, sodass nicht der Autor, sondern nur der Verein selbst angezeigt wird. Ist die Anzeige des Autors aktiviert, kann über das Brief-Symbol (links vom Namen) ein neuer Chat mit dem Verfasser gestartet und eine private Nachricht gesendet werden, sofern man nicht selbst der Verfasser ist (Siehe Abbildung 8).

5.3 Verwaltung

Für Vereinsbetreiber gibt es einen zusätzlichen Bildschirm (Siehe Abbildung 13). Hier werden alle Vereine angezeigt, bei denen der Nutzer als Administrator eingetragen ist. Oben rechts findet sich der Menüpunkt „Verein beitreten“, dieser wird bei Vereinsmitgliedern direkt unten im Menü anstelle des zweiten Symbols angezeigt. Druch Klicken auf einen bestimmten Verein gelangt man zur Verwaltungsansicht. In der verwaltungsansicht können Anpassungen am Verein vorgenommen werden, wie zum Beispiel Name, Logo oder Farbe (Siehe Abbildung 10). Zudem können Dateien hochgeladen, Events oder Gruppen erstellt und QR-Codes (Siehe Abschnitt 6.2) generiert werden. Die Ansichten

der einzelnen Menüpunkte sind ähnlich, daher wird im Folgendem exemplarisch auf die Gruppen-Verwaltung eingegangen.

5.3.1. Gruppen-Verwaltung

Durch Gruppen können Vereinsbetreiber Mitteilungen gezielter an verschiedene Mitglieder des Vereins senden (Siehe Abschnitt 6.3). Gruppen können sowohl öffentlich zugänglich als auch privat sein. Während des Beitretens wählt der Benutzer aus den öffentlichen Gruppen aus (Siehe Abschnitt 6.2). Privaten Gruppen kann nur per Einladungscode und Bestätigung durch den Vereinsbetreiber beigetreten werden (Siehe Abschnitt 7.3.2). In der Gruppen-Verwaltung können bestehende Gruppen bearbeitet und neue erstellt werden. Zudem wird der Benutzer gewarnt, sobald Sicherheitsrisiken vorliegen (Siehe Abbildung 12).

5.4 Verein beitreten

Auf dem „Beitreten“-Screen können sowohl Vereinsbetreiber als auch Mitglieder einem neuen Verein beitreten (Siehe Abbildung 11). Durch die Suche kann einem öffentlichen Verein beigetreten werden. Verfügt der Nutzer über einen Einladungscode, so kann er dem Verein beitreten, auch wenn dieser nicht öffentlich ist. Vereinsbetreiber können Einladungscodes auch in Form von QR-Codes anbieten. Diese lassen sich durch einen Klick auf das Kamera-Symbol einscannen (Siehe Abbildung 11, oben rechts) . Geplant ist auch, dass QR-Codes direkt mit der vorinstallierten Kamera-App eingescannt werden können. Per sogenannten „Deeplinks“ könnte dann direkt zur App weitergeleitet und der zugehörige Verein geöffnet werden. Nachdem ein Verein ausgewählt, beziehungsweise ein QR-Code eingescannt wurde, kann der Nutzer Gruppen auswählen, denen er beitreten möchte. Gruppen die mit dem eingegeben oder eingscannten Einladungscode verknüpft sind, werden automatisch ausgewählt. Sie lassen sich aber auch wieder abwählen.

5.5. Chat

Auf der Chatübersicht werden alle aktiven Chats angezeigt. Jeweils mit dem Namen und dem Profilbild des Chatpartners, der letzten Nachricht, und der vergangenen Zeit seit der letzten Nachricht (Siehe Abbildung 16). Durch Klicken auf einen Chat gelangt man auf die Chatansicht (Siehe Abbildung 15). Hier werden die letzten Nachrichten mit Uhrzeit angezeigt. Unterhalb des Chatverlaufs kann man seine Nachricht in das Textfeld eintippen und, mit einem Klick auf den Button daneben, verschicken. Tippt der Chatpartner eine Nachricht ein, so wird dies unterhalb vom Namen im Header angezeigt (Siehe Abschnitt 8.3). Durch Scrollen nach oben können weitere Nachrichten angezeigt werden. Um die Internetverbindung nicht zu stark auszulasten werden gepeicherte Nachrichten aus einer lokalen SQL-Datenbank ausgelesen (Siehe Abschnitt 7.3.1).

5.6. Einstellungen

In den Einstellungen kann der Nutzer sein Profilbild bearbeiten, seinen Nutzernamen ändern und seine Vereins-Gruppen bearbeiten oder einen Verein verlassen (Siehe Abbildung 14). Außerdem hat der Benutzer die Möglichkeit seinen Account zu löschen.

6. Abläufe

6.1 Anmeldung

Die Anmeldung in der App erfolgt anonym. Nach einer kurzen Einleitung über die wichtigsten Funktionen der App kann der Nutzer einen Benutzernamen wählen (Siehe Abbildung 18). Der Benutzername darf aus Sicherheitsgründen nicht länger als 50 Zeichen sein, und keine Sonderzeichen enthalten. Benutzernamen können mehr als ein Mal vergeben werden. Der Nutzer wird bereits beim Starten der App anonym in Firebase angemeldet und erhält eine eindeutige Id. Klickt der Nutzer auf den Button „Fertig“ (Siehe Abbildung 20, unten rechts) wird der Benutzer-Account mit der zuvor zugewiesenen Id in der Firebase Datenbank erstellt. Die Id wird im lokalen Speicher gesichert, um beim

nächsten Start wieder drauf zugreifen zu können. Der angegebene Benutzername wird in der Datenbank gespeichert. Die von OneSignal generierte Id wird an die Datenbank übergeben. Zudem werden die öffentlichen und privaten Schlüssel zur Verschlüsselung der Chatnachrichten generiert (Siehe Abschnitt 7.3). Der öffentliche Schlüssel wird in der Datenbank gespeichert. Der private Schlüssel wird bei Android in „EncryptedSharedPreferences“ und bei IOS in „Keychain“ gespeichert (vgl. GITHUB – REACT NATIVE ENCRYPTED STORAGE, 2021, URL). Beides sind vom System zur Verfügung gestellte Möglichkeiten um sensible Daten sicher zu speichern. Nachdem der Benutzer registriert ist, wird er zum Startbildschirm weitergeleitet (Siehe Abschnitt 5.1). Da zu diesem Zeitpunkt noch keinem Vereinen begetreten wurde, werden auf der Startseite auch keine Events oder Mitteilungen angezeigt. Statdessen sieht der Nutzer eine kurze Erklärung (Siehe Abbildung 17).

6.2 Beitreten eines Vereins

Alle Nutzer können einem Verein betreten, hier gibt es keine Unterscheidung zwischen Vereinsbetreibern und Vereinsmitgliedern. Der Beitritt geschieht über den Beitreten-Screen (Siehe Abbildung 11). Öffentliche Vereine können per Suche gefunden werden, privaten Vereinen kann nur per Einladungscode beigetreten werden. Einladungscodes bestehen aus einer sechstelligen zufälligen Zahlen und Buchstaben Kombination. Die Codes können manuell in das Suchfeld eingegeben werden, oder als QR-Code eingescannt werden. Die QR-Codes werden in der App von den Vereinsbetreibern zusammen mit den Einladungscodes generiert. Und können, zum Beispiel im Vereinsgebäude, aufgehangen werden. So können neue Mitglieder unkompliziert beitreten. Die Codes sind mit dem Verein selbst, und mit vorher ausgewählten Gruppen verknüpft. So hat man die Möglichkeit neue Mitglieder direkt einer (privaten) Gruppe zuzuweisen. Die Verwendung von Einladungscodes ist für den Zugriff auf private Vereine und Gruppen unumgänglich. Denn der Vereinsbetreiber kann Nutzer nicht manuell dem Verein oder einer Gruppe hinzufügen, da eine eindeutige Identifizierung der Nutzer durch die anonymisierung nicht möglich ist. QR-Codes können auf dem Beitreten-Screen mit dem Kamera-Symbol

eingescannt werden (Siehe Abbildung 11, oben rechts). Nach dem Klick auf das Symbol, öffnet sich der Scanner und der QR-Code muss vor die Kamera des Gerätes gehalten werden. Wird ein QR-Code erkannt, so ist der Ablauf gleich wie beim Aufruf des Vereins über das Suchfeld. Es öffnet sich ein Fenster, in dem der Nutzer die jeweiligen Gruppen auswählt denen er beitreten möchte. Gruppen, die mit dem erkannten Code verknüpft sind, werden automatisch ausgewählt. Sie können vom Nutzer aber auch wieder abgewählt werden. Private Gruppen werden nur angezeigt, sofern sie mit dem Einladungscode verknüpft sind. Mit einem Klick auf „Fertig“ werden die beigetretenen Gruppen im Benutzer-Object in der Firebase Datenbank gespeichert. Zudem wird der entsprechende OneSignal-Tag gesetzt, damit die Push-Notifications empfangen werden können (Siehe Abschnitt 3.3). Möchte der Nutzer einer privaten Gruppe beitreten, so wird dies nicht sofort wirksam. Zunächst wird eine Push-Nofication an die Vereinsbetreiber versendet, welche dann das neue Mitglied bestätigen, oder ablehnen können (Siehe Abschnitt 7.3.2). Die Mitteilungen und Events des Vereins werden nun auf der Startseite angezeigt.

6.3 Senden einer Mitteilung

Mitteilungen an die Teilnehmer gehören zu den Grundfunktionen der App und können ausschließlich von Vereinsbetreibern versendet werden. Geplant ist auch ein Webinterface über das Mitteilungen versendet und Dateien hochgeladen werden können. Auch bereits genannte Einstellungen zum Anzeigen des Autors bei Mitteilungen werden im Webinterface verfügbar sein.

Um eine neue Mitteilung zu erstellen, klickt man unten in der Navigationsleiste auf das Plus Symbol. Dadurch öffnet sich ein neues Fenster. Hier wird dann der Verein ausgewählt, an welchen die Mitteilung gesendet werden soll (Siehe Abbildung 20). Durch den Button unten rechts im Bild lässt sich auf die nächste Seite navigieren. Im nächsten Schritt gibt der Vereinsbetreiber die Überschrift, einen kurzen Anreißer und den Text der Mitteilung ein (Siehe Abbildung 21). Danach können Veranstaltungen erstellt und Dateien hochgeladen werden, die mit der Mitteilung verknüpft werden. Geplant ist, dass hier auch auf bereits erstellte Inhalte des Vereins zurückgegriffen werden kann. Auf der darauf folgenden Seite kann ein Beitragsbild hochgeladen werden. Im letzten Schritt

wählt der Betreiber die Gruppen aus, an die die Mitteilung gesendet werden soll. Hier kann aus den bestehenden Gruppen ausgewählt werden. Zusätzlich können Gruppen, durch einen Klick auf das Link Symbol, miteinander verlinkt werden (Siehe Abbildung 22). Bei miteinander verlinkten Gruppen wird die Mitteilung nur an die Mitglieder gesendet, die gleichzeitig in beiden Gruppen angemeldet sind. Anhand eines Beispieles wird diese Funktionsweise deutlicher: Der Vereinsbetreiber hat zwei Gruppen eingerichtet. Eine Gruppe „Freizeit 2021“ in der sowohl Mitglieder als auch Mitarbeiter dieser Freizeit angemeldet sind. Zudem gibt es eine Gruppe „Mitarbeiter“ in der alle Mitarbeiter des Vereins gelistet sind. Nun soll eine Mitteilung gesendet werden, allerdings nur an alle Mitarbeiter der „Freizeit 2021“. Durch das Verknüpfen beider Gruppen werden genau die gewünschten Mitglieder erreicht. Da sie zum Empfangen der Mitteilung sowohl in der Gruppe „Freizeit 2021“ als auch in der Gruppe „Mitarbeiter“ gelistet sein müssen. Somit kann auf unnötige Gruppen wie „Mitarbeiter Freizeit 2021“ verzichtet werden.

Nachdem die Inhalte und Empfänger der Mitteilung bestimmt wurden, klickt man auf das Senden-Symbol unten rechts im Bildschirm (Siehe Abbildung 22). Die Mitteilung wird nun in der Firebase Datenbank gespeichert. Auf diese Änderung reagiert der Datenbank-Trigger „sendMessageNofitications“ in den Firebase Functions:

```
1. exports.sendMessageNotifications =
2. functions.database.ref('clubs/{club_id}/messages/{mes_id}').onCreate(async (snapshot, context) (..)
```

Dieser Trigger wird ausgeführt, sobald ein neues Child Object unter „messages“ im Club-Object erstellt wird. Der Firebase Server sendet nun mithilfe von OneSignal die Push-Notification an die Nutzer, die in den (verlinkten) Gruppen eingetragen sind. Für die jeweiligen Vereinsmitglieder erscheint nun eine neue Mitteilung auf dem Startbildschirm der App.

7. Datensicherheit

Einer der Gründe für das Nutzen der App sollte die Datensicherheit sein, welche von alternativen Anbietern nicht angeboten wird. Dieser Gedanke wird durch zwei Grundprinzipen umgesetzt. Erstens so wenig Daten wie möglich über

Nutzer und Vereine zu sammeln, und zweitens die geringe Menge an Daten so sicher wie möglich zu speichern. Bei werbefinanzierten Prdukten wie zum Beispiel Facebook wäre diese Strategie undenkbar, da durch die gesammelten Information gezielter Anzeigen geschaltet werden können (vgl. HEISE ONLINE – WIE FACEBOOK MIT IHREN DATEN GELD VERDIENT, 2012, URL).

7.1 Anonymes Verwenden der App

Die App kann daher als Vereinsmitglied komplett anonym verwendet weren, also ohne Regrestierung mit E-Mail-Adresse oder Telefonnummer. Dabei ergibt sich jedoch das Problem, dass die Nutzerdaten verloren gehen, sobald die App wieder deinstalliert wird. Dieses Problem besteht vorallem für Vereinsbetreiber, die bei einer Neu-Installation der App keinen Zugriff mehr auf ihren Verein hätten. Daher wird bei der Erstellung eines eigenen Vereins zunächst ein herkömmliches Konto mit E-Mail Adresse und Passwort erstestellt werden müssen.

7.2 Firebase

Das Firebase Backend wird von Google selbst vor unbefugten Zugriffen geschützt. Auf den gesamten Inhalt der Datenbank und auf die Firebase Functions lässt sich nur mit dem Firebase Administrator Account zugreifen. Die Datenbank ist zudem durch die sogenannten „Realtime Database Rules“ geschützt. Normalerweise kann jeder angemeldete Benutzer alle Einträge der Datenbank lesen und verändern. Für dieses Projekt ist eine solche Einstellung jedoch nicht ausreichend. Durch die Anoyme Registrierung ist es möglich ohne jegliche Verifizierung auf die Datenbank zuzugreifen, was ein hohes Sicherheitsrisiko darstellen würde, wenn die Datenbank nicht ausreichend gesichert wäre. Die „Realtime Database Rules“ für dieses Projekt sind beispielsweise so eingestellt, dass jeder Nutzer nur in sein eigenes Nutzer-Objekt schreiben kann. Zudem ist es, bis auf Bentutzernamen und Profilbild, nicht möglich Daten anderer Nutzer auszulesen. Auch die Daten von privaten Vereinen, können nicht eingesehen werden, sofern der Nutzer kein Mitglied dieses Vereins ist. Vereinsdaten können auch nur von Vereinsbetreibern

geändert werden. Außerdem ist das Hochladen von größeren Datenmengen, wie zum Beispiel sehr langen Zeichenketten, nicht möglich. Die „Realtime Database Rules“ werden im JSON-Format geschrieben und in Firebase gespeichert. Hier ein Beispiel für eine Datenbank-Regel:

```
{
        "rules":  {
                "users": {
                        "$uid": {
                                ".write": "$uid === auth.uid"
                        }
                }
        }
}
```

Auf das Benutzer-Objekt werden nur Schreibzugriffe vom Benutzer selbst zugelassen. „uid“ ist hierbei die ID des Benutzers.

7.3 Ende-Zu-Ende Verschlüsselung

Die Ende-Zu-Ende Verschlüsselung ist eines der Hauptbestandteile des Datenschutzkonzepts. In der App werden sowohl private Chat Nachrichten als auch Mitteilungen, die ausschließlich an private Gruppen gerichtet sind, verschlüsselt übertragen (Siehe Abschnitt 7.3.2). In der Praxis heißt dies, das der Sender die Nachricht verschlüsselt, bevor sie in die Datenbank gespeichert wird. Die Nachricht ist dann selbst für den Systemadministrator nicht entschlüsselbar. Auch Zwischeninstanzen (Man-in-the-Middle-Angriff), die den Internetverkehr mitlesen, können die Nachricht nicht entschlüsseln. Somit sind persönliche Daten und Chatverläufe auch bei einem direkten Angriff auf die Datenbank geschützt. Für dieses Projekt würde das RSA-Verfahren verwendet. Das RSA-Verfahren ist ein asymmetrisches kryptografisches Verfahren, das zum Verschlüsseln von zum Beispiel Chat-Nachrichten verwendet werden kann.

7.3.1 Der Weg einer Chatnachricht

Genauer lässt sich das RSA-Verfahren anhand von Chatnachrichten erklären. Wichtig für die Verschlüsselung nach dem RSA-Verfahren sind der öffentliche

Schlüssel (Siehe Abbildung 23, „Public Key“) und der private Schlüssel (Siehe Abbildung 23, „Private Key“). Dieses Schlüsselpaar wird bei der Einrichtung der App automatisch auf dem Gerät generiert. Der öffentliche Schlüssel wird dann für jeden zugänglich in der Datenbank gespeichert. Mit diesem Schlüssel kann keine Nachricht entschlüsselt werden, er dient nur zur Verschlüsselung. Vergleichbar ist der öffentliche Schlüssel mit einem Schloss, welches Gegenstände zwar abschließen kann, aber ohne passenden Schlüssel nicht geöffnet werden kann. Der private Schlüssel dient nun zur Entschlüsselung der Nachricht, also zum Aufschließen des Schlosses. Dieser wird sicher auf dem Gerät des Eigentümers gespeichert. Der private Schlüssel kann, anders als bei einem Schloss, nicht vom öffentlichen Schlüssel hergeleitet werden.

Wird von Benutzer 1 eine Nachricht an Benutzer 2 gesendet, so muss der Sender zunächst den öffentlichen Schlüssel des Empfängers aus der Firebase Datenbank abfragen (Siehe Abbildung 24, Übergang 1). Der öffentliche Schlüssel für den jeweiligen Benutzer ist in der Firebase Datenbank unter „users/[user_id]/public_key“ gespeichert. Die noch lesebare Nachricht wird dann mit dem öffentlichen Schlüssel des Empfängers verschlüsselt (Siehe Abbildung 24, „Nachricht mit public_key verschlüsseln“). Von Benutzer 1 wird nun die verschlüsselte Nachricht in die Firebase Datenbank hochgeladen (Siehe Abbildung 24, Übergang 2). Außerdem wird die nicht verschlüsselte Nachricht in der lokalen SQL-Datenbank gesichert, um später schneller darauf zugreifen zu können und sie aus der Firebase Datenbank löschen zu können. Die Methode „sendChatMessageNotifications“ in den Firebase Functions reagiert nun auf die hochgeladene Nachricht und sendet per OneSignal eine Push-Notification an den Empfänger (Siehe Abbildung 24, Übergang 3 und 4).

Öffnet der Benutzer 2 nun die App, werden die neuen Nachrichten aus der Datenbank abgefragt (Siehe Abbildung 25, „Neue Nachrichten abfragen“). Die Datenbank gibt dann die verschlüsselte Nachricht an den Empfänger zurück. Um die Nachricht zu entschlüsseln, benötigt es den privaten Schlüssel des Empfängers. Dieser ist im EncryptedStorage gespeichert und kann über den KeyManager ausgelesen werden. Der Schlüssel wird also nun aus dem EncryptedStorage ausgelesen (Siehe Abbildung 25, „EMFPÄNGER KEYMANAGER“). Mit dem privaten Schlüssel wird nun die Nachricht

entschlüsselt. Um größere Mengen an Nachrichten schnell laden zu können wird die entschlüsselte Nachricht in eine lokale SQL-Datenbank gespeichert. Da sowohl Sender als auch Empfänger über eine lokale Kopie der Nachricht verfügen, kann sie aus der Firebase Datenbank gelöscht werden. Es befinden sich also immer nur verschlüsselte Nachrichten in der Firebase Datenbank, die noch nicht vom Empfänger gepspeichert wurden. Haben sowohl Sender und Emfpänger das Chatfentster geöffnet, so befinden sich die verschlüsselten Nachrichten meist nur für 1-2 Sekunden in der Firebase Datenbank, bevor sie gelöscht werden.

7.3.2 Gruppenmitteilungen

Die Ende-Zu-Ende-Verschlüsselung gestaltet sich bei Gruppen-Mitteilungen schwieriger. In der App sind ausschließlich Mitteilungen privater Gruppen Ende-Zu-Ende verschlüsselt, da öffentlichen Vereinen und Gruppen ohnehin jeder beitreten kann. Zudem wäre die Verschlüsselung von Mitteilungen, mit potentiell unendlich vielen Empfängern, deutlich aufwänder.

Um Gruppen-Mitteilungen zu verschlüsseln wird beim Erstellen einer privaten Gruppe ein Schlüsselpaar aus privatem und öffentlichem Schlüssel erstellt. Der öffentliche Schlüssel kann, wie bei Benutzern auch, in der Firebase Datenbank gespeichert werden. Würden privater und öffentlicher Schlüssel zusammen in der Datenbank gespeichert werden, so wäre die Ende-Zu-Ende-Verschlüsselung unwirksam, da der private Schlüssel von Angreifern ausgelesen werden könnte. Daher wird für den privaten Schlüssel von privaten Gruppen auf die selbe Speichermethode wie für private Schlüssel von Benutzern zurückgegriffen: Dem EncryptedStorage. Auf den EncryptedStorage kann über die KeyManager-Klasse zugegriffen werden. Beim Erstellen einer privaten Gruppe wird der öffentliche Schlüssel also in der Firebase Datenbank und der private Schlüssel der Gruppe im EncryptedStorage gespeichert. Das Problem hierbei ist jedoch, dass bisher nur der Ersteller der Gruppe selbst Zugriff auf den privaten Schlüssel hat. Um die Mitteilungen entschlüsseln zu können bräuchte jedes Mitglied der Gruppe Zugriff auf den privaten Schlüssel. Der folgende Ablauf einer Anmedlung in einer privaten Gruppe dient also primär

dazu den zuvor generierten privaten Schlüssel an das neue Gruppen-Mitglied weiterzuleiten.

Möchte ein Nutzer einer privaten Gruppe beitreten wird dieser zunächst mit der Benutzer Id einer Warteschlange im Gruppen-Objekt des Vereins in der Datenbank hinzugefügt (Siehe Abbildung 26, Übergang 1). Auf das neue Element in der Warteschlange reagiert nun der Trigger in den Firebase Functions (Siehe Abbildung 26, Übergang 2). Daraufhin sendet die Funktion eine Push-Notification über den OneSignal Service an den Vereinsbetreiber (Siehe Abbildung 26, Übergang 3). Der Betreiber erhält nun eine Benachrichtigung mit dem Inhalt „Benutzer X möchte der Gruppe Z beitreten. Anfrage bestätigen?“. Solange diese Mitteilung nicht geöffnet wird, passiert erstmal nichts.

Sobald der Vereinsverwalter dann die Push-Nofitication öffnet, startet sich die App. Hier wird dann nochmals die selbe Nachricht in einem Auswahlfenster angezeigt (Siehe Abbildung 27, „Anfrage als Popup anzeigen“). Der Nutzer hat die Auswahl zwischen „Ja“ und „Nein“. Beim Klicken auf „Nein“ wird die Anfrage aus der Datenbank gelöscht und das Auswahlfenster schließt sich (Siehe Abbildung 27, Übergang 1 zu Firebase Database). Klickt der Nutzer auf „Ja“ (Siehe Abbildung 27, Übergang 1) so wird der private Schlüssel der Gruppe aus dem EncryptedStorage ausgelesen (Siehe Abbildung 27, Übergang 2). Im drauf folgendem Schritt wird der öffentliche Schlüssel des Mitglieds aus der Firebase Datenbank abgefragt (Siehe Abbildung 27, Übergang 3). Mit dem öffentlichen Schlüssel wird nun der private Gruppen-Schlüssel verschlüsselt (Siehe Abbildung 27, „private_group_key mit public_key verschlüsseln“). Den verschlüsselten Schlüssel kann außschließlich das neue Mitglied entschlüsseln. Dieser wird dann in die Warteschlange der Gruppe in der Firebase Datenbank hochgeladen (Siehe Abbildung 27, Übergang 4). Auf diese Änderung in der Datenbank reagiert daraufhin ein Trigger in den Firebase Functions (Siehe Abbildung 27, Übergang 5). An das neue Mitglied wird per OneSignal eine Push-Notification gesenedet. Mit dem Inhalt „Deine Anfrage für die Gruppe X wurde angenommen“.

Klickt der Nutzer nun auf die Push-Notification oder öffent die App (Siehe Abbildung 28, Übergang 1) wird der eigene private Schlüssel aus dem EncryptedStorage ausgelesen (Siehe Abbildung 28, Übergang 2). Zudem wird der verschlüsselte private Gruppen-Schlüssel aus der Firebase Datenbank abgefragt (Siehe Abbildung 28, Übergang 3). Nun wird der verschlüsselte Gruppen-Schlüssel mit dem eigenen privaten Schlüssel entschlüsselt (Siehe Abbildung 28, „encrypted_private_group_key mit private_key entschlüsseln). Daraufhin wird der entschlüsselte private Gruppen-Schlüssel im EncryptedStorage gespeichert (Siehe Abbildung 28, Übergang 4). Somit kann dieser beim Empfang einer Mitteilung aus der privaten Gruppe ausgelesen werden und die Mitteilung entschlüsselt werden. Der Beitritt der Gruppe wird nun in der Firebase Datenbank verzeichnet (Siehe Abbildung 28, Übergang 5) und die Anfrage wird aus der Datenbank gelöscht (Siehe Abbildung 28, Übergang 6).

Dieses Verfahren zum Austausch des privaten Gruppen-Schlüssels ist technisch aufwendig, und geschieht über einen kleinen Umweg. Es bietet jedoch die größtmögliche Datensicherheit für die Übermittlung privater Gruppenmitteilungen. Außerdem behält der Vereinsverwalter die Entscheidungsgewalt darüber, welcher Nutzer seiner privaten Gruppe beitritt und wer nicht.

8. Besondere Stellen im Code

8.1 DatabaseConnector.js

Die DatabaseConnector-Klasse ist eine selbst erstellte Schnittstelle zur Firebase Datenbank. Sie erleichtert die Implementation von Unterklassen, die auf Objekte in der Firebase Datenbank zugreifen müssen, welche mehre Child-Objekte mit eindeutigen Ids besitzen. Dies sind zum Beispiel das „events“-Objekt, welches für jeden Verein Veranstaltungen mit je einer eindeutigen Id listet. Von dem DatabaseConnector unterstütze Pfade müssen daher folgendermaßen strukturiert sein. Hier ein Beispiel als JSON-Objekt:

```
1.  {
```

```
2.          "referenz_name": {
3.                  "child_id": {
4.                          "key":"value"
5.                  },
6.                  "child_id": {
7.                          "key":"value"
8.                  }
9.          }
10. }
```

Der Zugriff auf die lokale SQL-Datenbank ist auch über den DatabaseConnector möglich.

In React Native ist der Zugriff auf die Firebase Datenbank über das „react-native-firebase“ Paket möglich. Um auf einen Pfad zuzugreifen muss zunächst eine Datenbankreferenz zu dem Pfad erzeugt werden. Dies ist möglich mit:

```
const ref = database().ref(path)
```

Mit dieser Datenbankreferenz kann nun ein sogennanter „Listener“ erzeugt werden. Die Firebase Listener sind verknüpft mit der Datenbankrefernz und führen die hinterlegte Callback-Funktion aus, sobald sich die Daten der Referenz in der Datenbank ändern. Somit lässt sich in Echtzeit auf Vorgänge in der Datenbank reagieren. Die Listener lassen sich verwenden, um die Daten einmalig aufzurufen („once“) oder auf jede Änderung zu reagieren („on“).

Hier ein Beispiel für einen Firebase-Listener, der auf jede Änderung im zuvor festgelegten Pfad reagiert:

```
1. ref.on("value", function(snap) {
2.         console.log(snap.val())
3. }.bind(this));
```

Bei der Ausführung der Callback-Funktion liefert der Listener das Objekt „snap“ als Parameter. Von „snap.val()“ werden dann die Daten der Datenbank als JSON-Objekt zurückgegeben.

Der Konstruktor von DatabaseConnector erwartet zwei notwendige Parameter. Dies sind „parent_path“ und „id“. Zusätzlich kann auch ein Array unter „start_values“ und ein JSON-Objekt unter „data“ angegeben werden. In der vorhin angegebenen Stukturierung entspricht „parent_path“ dem Pfad zum Objekt „referenz_name“ und der Parameter „id“ der Id des Childs, also

„child_id“. Die Anfragen an die Datenbank gehen nun an das Objekt in „parent_path/id“. Durch den Parameter „data“ kann das Child-Objekt übergeben werden, falls es bereits vorher aus der Datenbank geladen wurde. Mit dem Parameter „start_values“ können Pfade als Array angegeben werden, die heruntergeladen sein sollen, bevor der „readyListener“-Callback ausgeführt wird, dazu später mehr. Im DatabaseConnector können die Firebase-Listener mit der Funktion „startListener“ gesetzt werden. Hier muss als Parameter die Callback-Funktion angegeben werden, die aufgerufen wird sobald die Daten heruntergeladen oder verändert wurden. Als zweiter Parameter werden ein oder mehrere Pfade angegeben, auf die der Listener reagieren soll. Für einen Pfad können bei Bedarf mehrere Listener erstellt werden. Im Konstruktor wird für jeden Pfad in „start_values“ ein Listener erstellt. Mit der Funktion „setReadyListener“ kann ein Listener erstellt werden, welcher ausgeführt wird, sobald das Objekt „bereit“ ist. Also sobald alle in „start_values“ angegebenen Werte geladen sind. Somit kann sichergestellt werden, dass beispielweise beim Rendern, keine erforderlichen Werte fehlen. In der Funktion „checkIfReady“ werden dann bei jeder Änderung alle „start_values“ überprüft und gegebenfalls der „ready“-Listener ausgeführt. Für ein DatabaseConnector-Objekt können mehere „ready“-Listener erstellt werden. Werden keine „start_value“ angegeben, oder befinden sich die Werte im als Parameter übergebenen „data“-Objekt, so wird der „ready“-Listener sofort nach Erstellen des Objekts ausgeführt. Mit „setValue“ können Werte im „data“-Objekt gespeichert werden, oder, sofern der Parameter „store“ auf „true“ gesetzt ist, direkt in die Datenbank geschrieben werden. Mit der Funktion „getValue“ können Werte aus dem „data“-Objekt zurückgegeben werden. Wird als Parameter eine Callback-Funktion bereitgesellt, so werden die Werte aus der Firebase Datenbank geladen und die Callback-Funktion ausgeführt. Zudem können per „removeValue“ Werte aus der Datenbank gelöscht werden. Mit „countUp“ kann ein Integer-Wert um eins hochgezählt werden. Die Funktion „hasValue“ gibt zurück, ob ein bestimmter Wert heruntergeladen wurde. Beziehungsweise, bei Angabe einer Callback-Funktion, in der Firebase Datenbank gespeichert ist. Mit „stopAllListener“ können alle Listener gestoppt und entfernt werden.

8.2 HeaderScrollView.js

Der HeaderScrollView ist eine Abwandlung vom React Native „ScrollView“. Die Klasse bietet die Möglichkeit eine Überschrift in ein scrollbare Ansicht (ScrollView) zu integrieren, und mit der Scollbewegung zu verknüpfen. Der HeaderScrollView wird auf fast allen Screens verwendet. Im Folgenden wird der HeaderScrollview anhand der Gruppen-Verwaltung erklärt.

Abbildung 3

Abbildung 2

Abbildung 4

Im Ausgangszustand wird die Überschrift in normaler Größe angezeigt (Siehe Abbildung 3). Sobald der Nutzer beginnt zu scrollen, wird langsam ein BlurView-Element eingeblentet, welches den Hintergrund unscharf macht (Siehe Abbildung 2). Es wird die selbe Überschrift in klein angezeigt, welche weniger Platz benötigt, und daher besser in die Kopfzeile passt (Siehe Abbildung 4). Wurde weit genug runter gescrollt, so verschwindet die große Überschrift hinter dem BlurView und die kleinere Überschrift ist komplett sichtbar (Siehe Abbildung 4). Neben der Animation der Überschrift können Navigations- und Aktionsbuttons eingeblendet und mit Callback-Funktionen ausgestattet werden (Siehe Abbildung 3, oben links und rechts). Außerdem kann unter der kleinen Überschrift eine weitere sogenannte „Subheadline“ eingeblendet werden. Dies kommt beispielsweise bei der Chatansicht zum Einsatz (Siehe Abbildung 5).

Abbildung 5

Der „ActionButton“ (Siehe Abbildung 3, oben rechts) kann außerdem bei Bedarf ausgeblendet werden (Siehe Abbildung 5).

Die Animationen der Überschriften und Hintergründe ist mit der Bewegung des ScrollViews verbunden. Sie laufen also parallel zur Scrollbewegung ab. Dies ist durch die React Native Animated API möglich. Die Animated API beinhaltet die gängigen React Native Elemente, wie zum Beispiel ein ScrollView. Die herkömmliche Einbindung eines ScrollViews, innerhalb der render-Methode, sieht so aus:

```
1. <ScrollView>
2. (...)
3. </ScrollView>
```

Um auf das ScrollView der Animated API zuzugreifen, muss das ScrollView folgendermaßen verändert werden:

```
1. <Animated.ScrollView
2. onScroll={this.getScrollCallback()}
3. >
4. (...)
5. </Animated.ScrollView>
```

Um auf die aktuelle Position des ScrollViews zuzugreifen, wird der Parameter „onScroll“ mit dem „ScrollCallback“ ausgestattet. Das ScrollCallback wurde für Testzweke in die Funktion „getScrollCallback“ ausgelagert. In der Regel erfolgt die Zuweisung aber direkt in der render-Methode. Dies ist die Implementation der Funktion „getScrollCallback“:

```
1. getScrollCallback() {
2.     return Animated.event([
3.       {
4.         nativeEvent: {
5.           contentOffset: {
6.             y: this.state.scrollY
7.           }
```

```
8.          }
9.        }
10.     ]).bind(this);
11.   }
```

Für die Animation von Elementen benötigt es eine „Animated.Value“. Im HeaderScrollView heißt diese Variable „scrollY“. ScrollY wurde im Konstruktor mit „new Animated.Value(0)“ erstellt, oder wahlweise als Parameter an das HeaderScrollView übergeben. Mit der Funktion „Animated.event()“ können nun die Gesten, wie zum Beispiel das Scrollen, direkt mit der Variable scrollY verbunden werden. ScrollY hat also immer die aktuelle Y-Koordinate des ScrollViews als Wert. In der render-Methode werden dann die zu animierenden Elemente mit scrollY verknüpft.

Dieser Vorgang wird im folgenden anhand der Header-Überschrift verdeutlicht. Die Header-Überschrift wird in der render-Methode erstellt:

```
1.  <Theme.Text
2.    style={{
3.              fontSize: 23,
4.              fontFamily: 'Poppins-Bold',
5.              marginTop: this._getHeaderHeadlineMarginTop(),
6.              opacity: this._getHeaderHeadlineOpacity()
7.    }}>
8.    {this.props.headline}
9.  </Theme.Text>
```

Es wird ein neues Objekt vom Typ „Theme.Text“ gerendert. Objekte aus der Klasse Theme sind an das aktuelle, dunkle oder helle, Design angepasst. Beim Text ändert sich beim Umschalten in den Dark Mode beispielsweise die Farbe von Schwarz nach Weiß Das Textelement in der Theme Klasse ist bereits vom Typ „Animated.Text“. Animiert werden im Style-Parameter der Abstand nach oben („marginTop“) und die Sichtbarkeit („opacity“). Die Werte dieser Parameter stammen aus den Methoden „_getHeaderHeadlineMarginTop“ für den Abstand nach oben und „_getHeaderHeadlineOpacity()“ für die Sichtbarkeit. Dies ist die Implementierung der Funktion „_getHeaderHeadlineMarginTop“:

```
1.  _getHeaderHeadlineMarginTop = () => {
2.    return this.state.scrollY.interpolate({
```

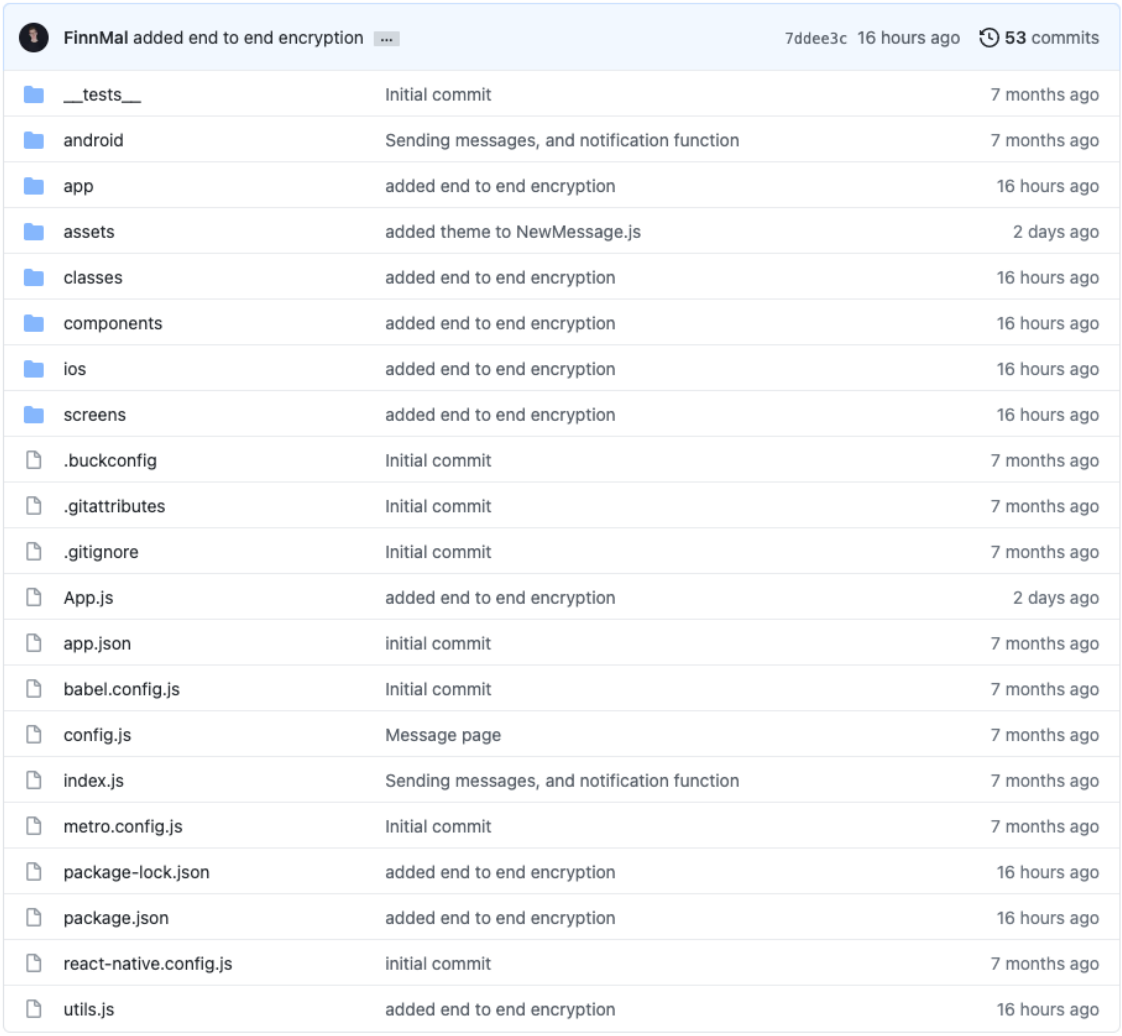
```
3.          inputRange: [
4.            -5, 50
5.          ],
6.          outputRange: [
7.            39, 4
8.          ],
9.          extrapolate: "clamp",
10.         useNativeDriver: true
11.       });
12.   };
```

Um die ScrollY zu verwenden muss die „interpolate“-Funktion aufgerufen werden. Damit können „inputRange“ und „outputRange“ sowie Parameter zur Animierung festgelegt werden. InputRange gibt an, ab welchem Wert von ScrollY die Animation starten beziehungsweise bei welchem Wert sie enden soll. Mit outputRange gibt man die Ausgabewerte für den Start und das Ende der Animation an. Wird die Animation also bei einem ScrollY Wert von -5 gestartet, so hat die Überschrift im Header zu diesem Zeitpunkt einen Abstand nach oben von 39 Pixeln. Die Werte werden bei jeder Änderung von ScrollY neu berechnet.

8.3 ScreenHandler.js

Der ScreenHandler verwaltet die Navigation über die Icons auf der Startseite. Nachdem die App in App.js gestartet wurde, wird auf den ScreenHandler-Screen gewechselt. Dieser besteht aus den Komponenten „HomeScreen“, „ManagmentScreen“, „AddClubScreen“, „MessagesScreen“ und „SettingsScreen“. Durch Klicken auf ein bestimmtes Icon werden alle anderen Screens ausgeblendet und nur der ausgewählte Screen angezeigt.

Abbildungsverzeichnis



FinnMal added end to end encryption ...	7ddee3c 16 hours ago	**53** commits
__tests__	Initial commit	7 months ago
android	Sending messages, and notification function	7 months ago
app	added end to end encryption	16 hours ago
assets	added theme to NewMessage.js	2 days ago
classes	added end to end encryption	16 hours ago
components	added end to end encryption	16 hours ago
ios	added end to end encryption	16 hours ago
screens	added end to end encryption	16 hours ago
.buckconfig	Initial commit	7 months ago
.gitattributes	Initial commit	7 months ago
.gitignore	Initial commit	7 months ago
App.js	added end to end encryption	2 days ago
app.json	initial commit	7 months ago
babel.config.js	Initial commit	7 months ago
config.js	Message page	7 months ago
index.js	Sending messages, and notification function	7 months ago
metro.config.js	Initial commit	7 months ago
package-lock.json	added end to end encryption	16 hours ago
package.json	added end to end encryption	16 hours ago
react-native.config.js	initial commit	7 months ago
utils.js	added end to end encryption	16 hours ago

Abbildung 6: Projektordner für die React Native App

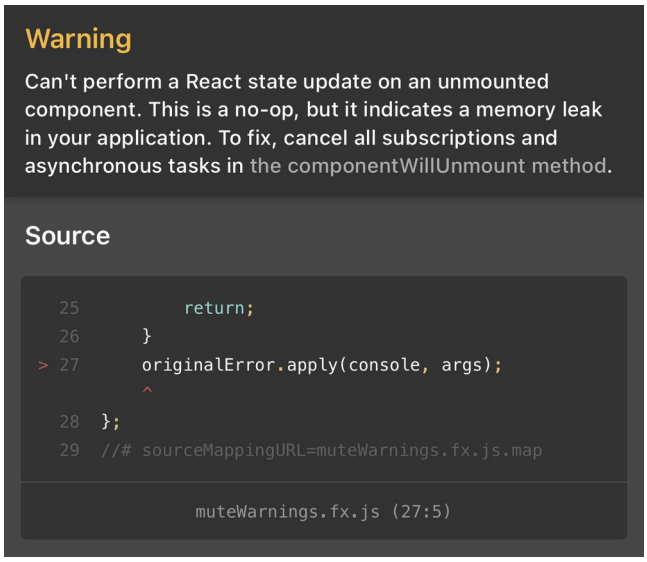


Abbildung 7

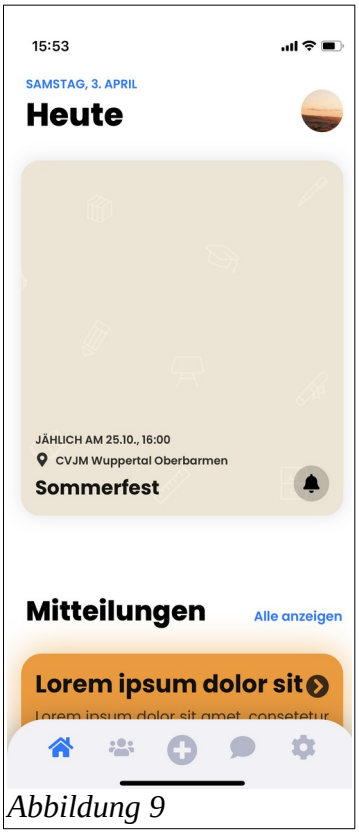


Abbildung 9

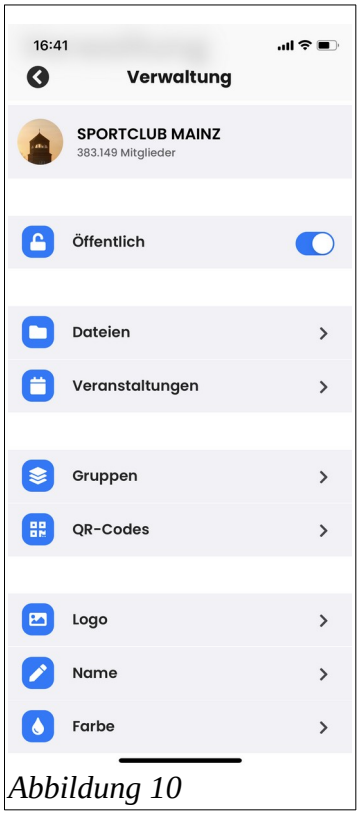


Abbildung 10

Abbildung 8

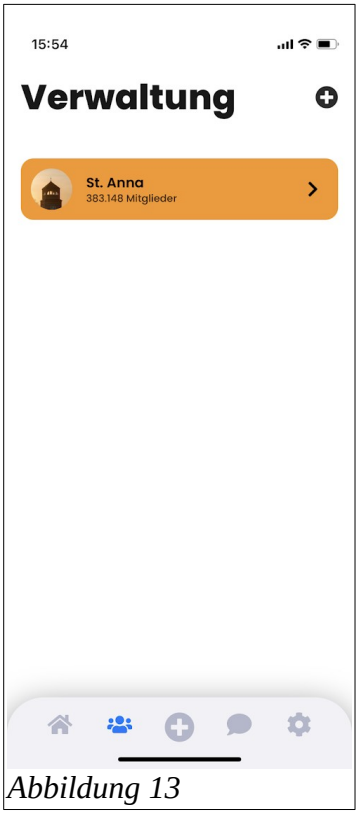


Abbildung 13

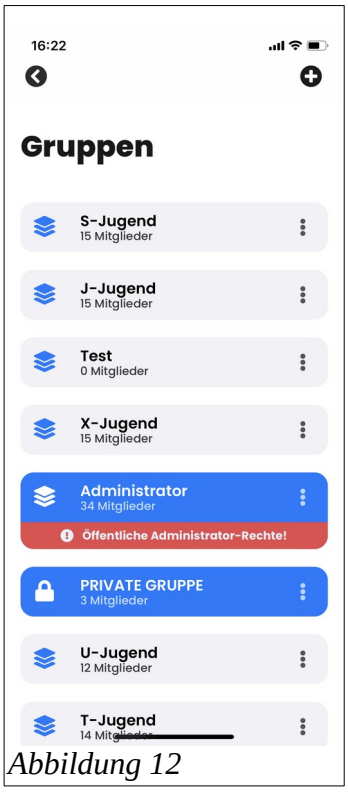


Abbildung 12

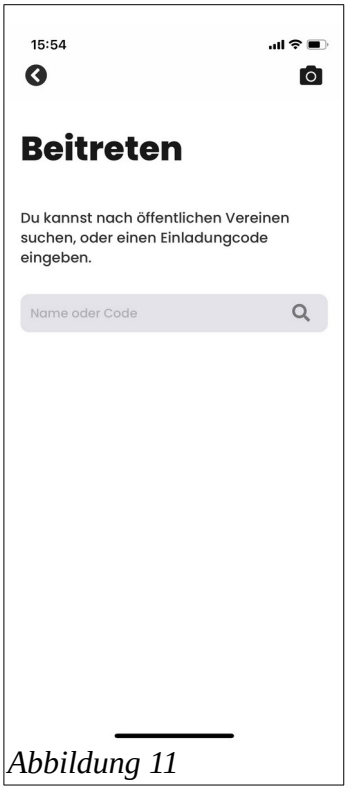


Abbildung 11

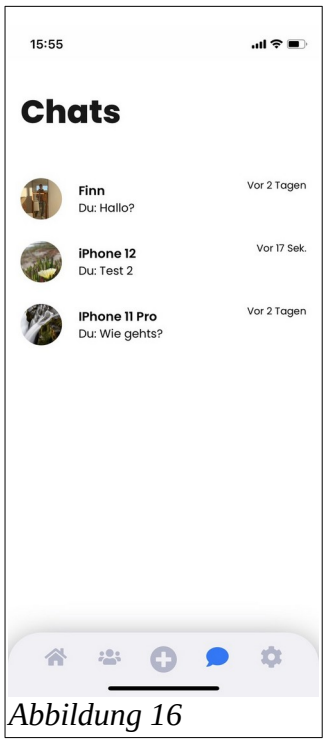


Abbildung 16

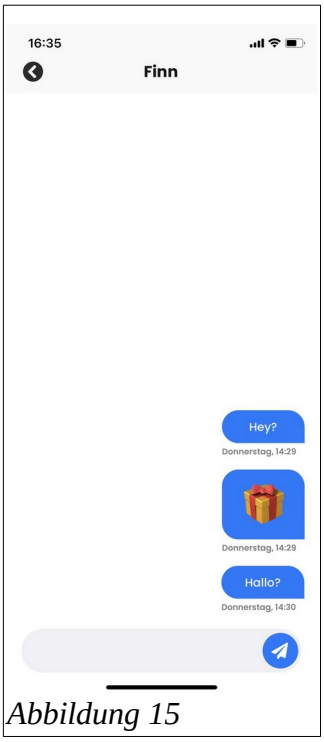


Abbildung 15

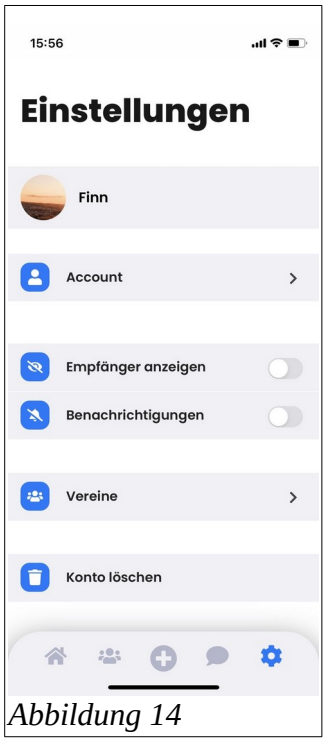


Abbildung 14

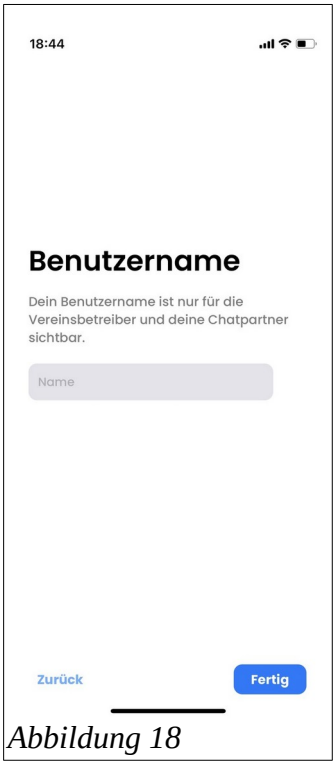


Abbildung 18

Abbildung 17

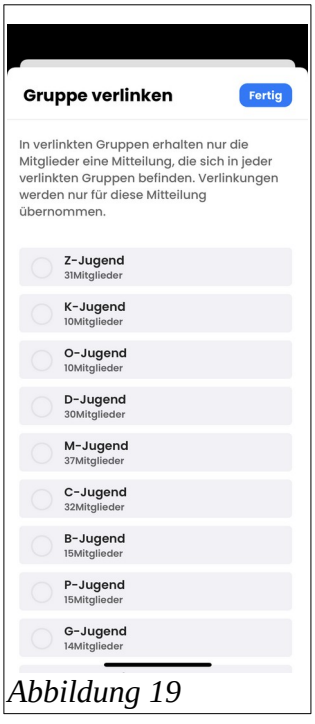


Abbildung 19

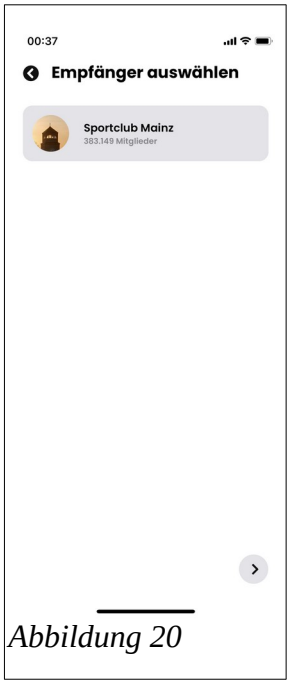


Abbildung 20

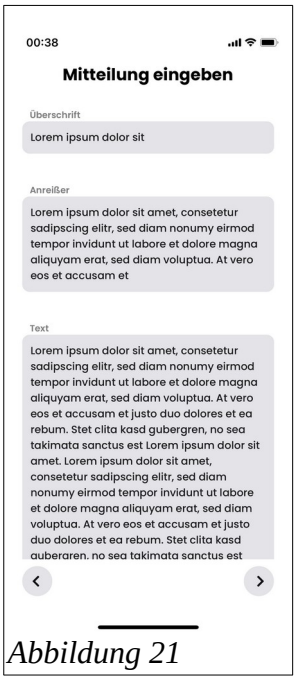


Abbildung 21

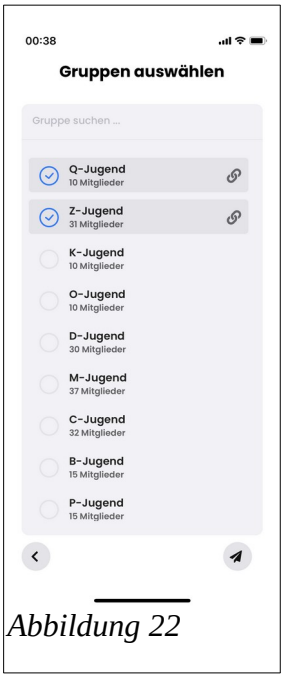


Abbildung 22

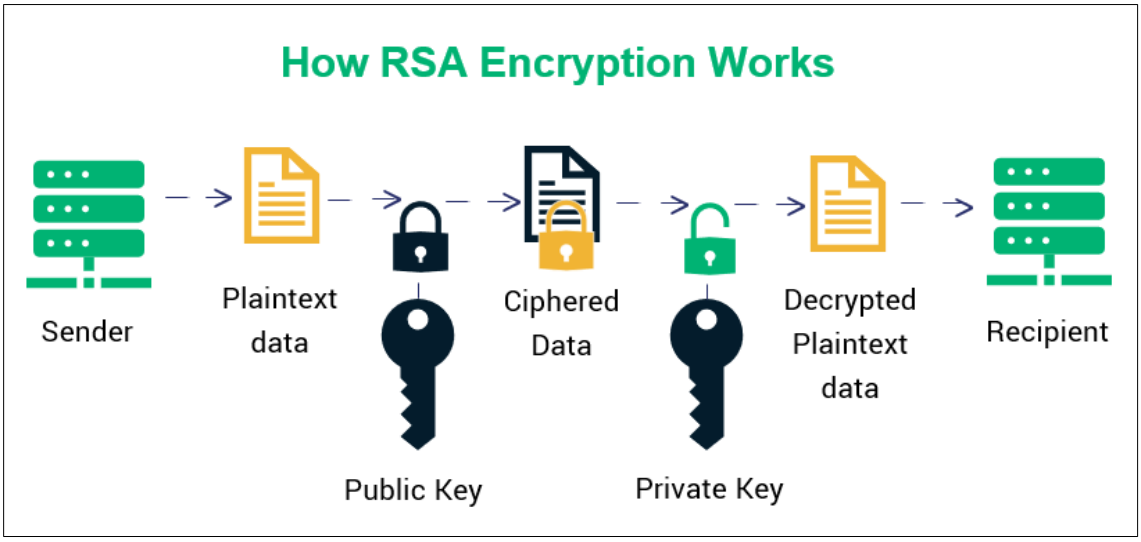


Abbildung 23: Ende-Zu-Ende Verschlüsselung

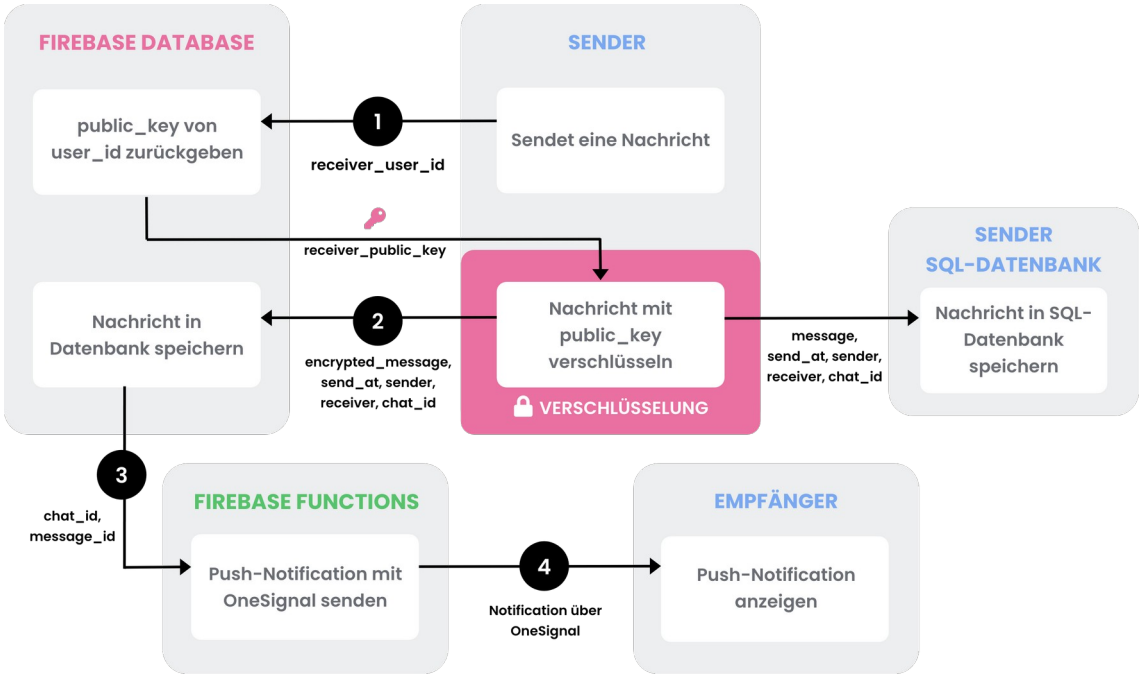


Abbildung 24: Der Weg einer Chat-Nachricht – Teil 1

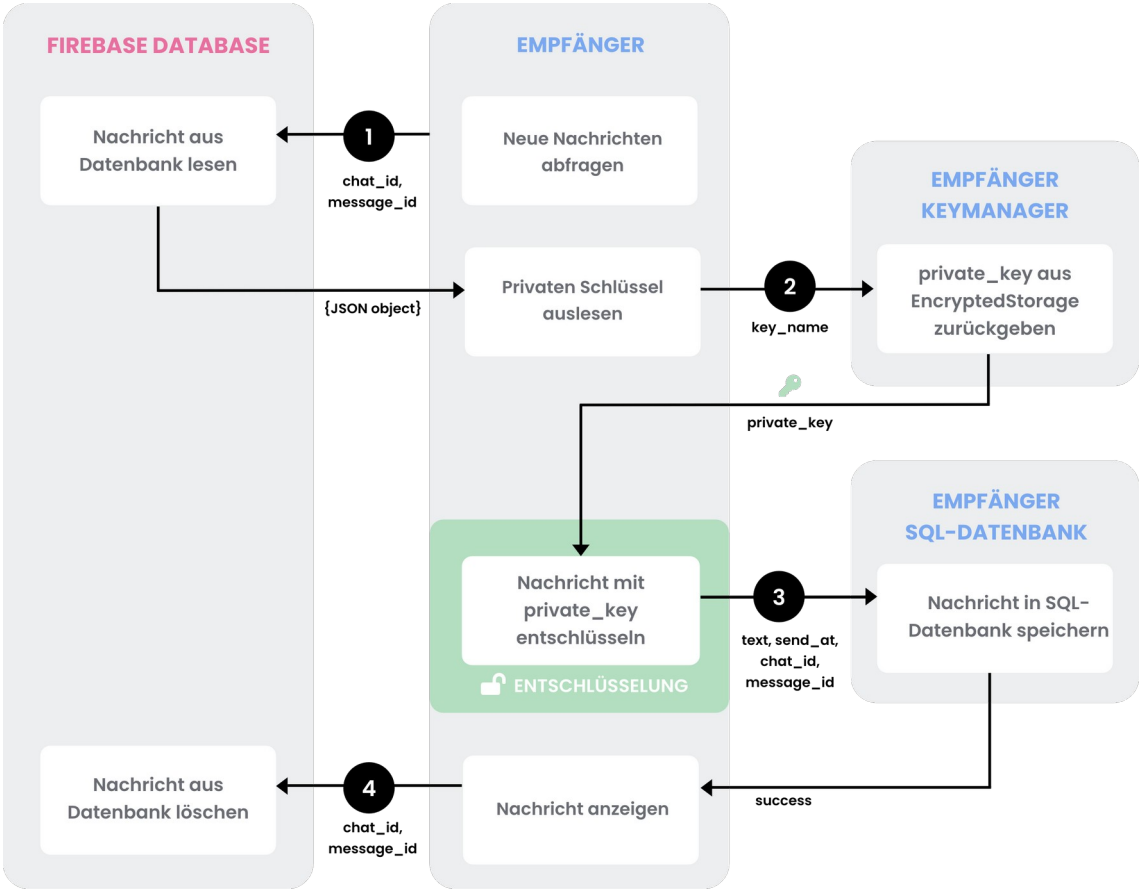


Abbildung 25: Der Weg einer Chat-Nachricht – Teil 2

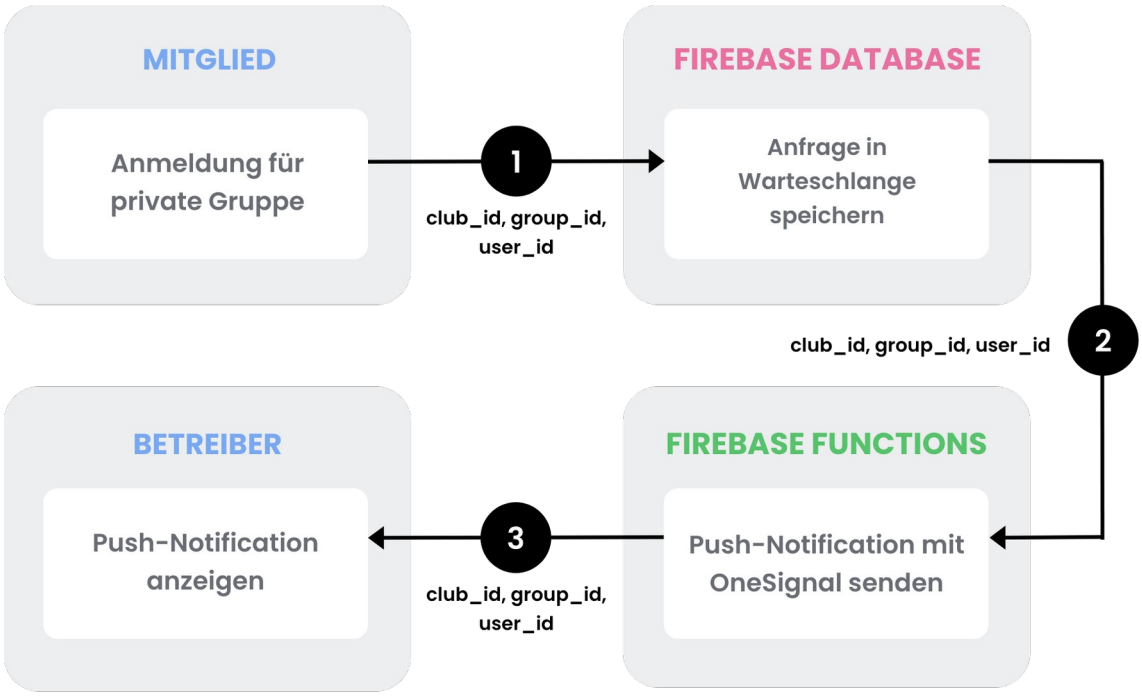


Abbildung 26: Austausch des privaten Gruppen-Schlüssels - Teil 1

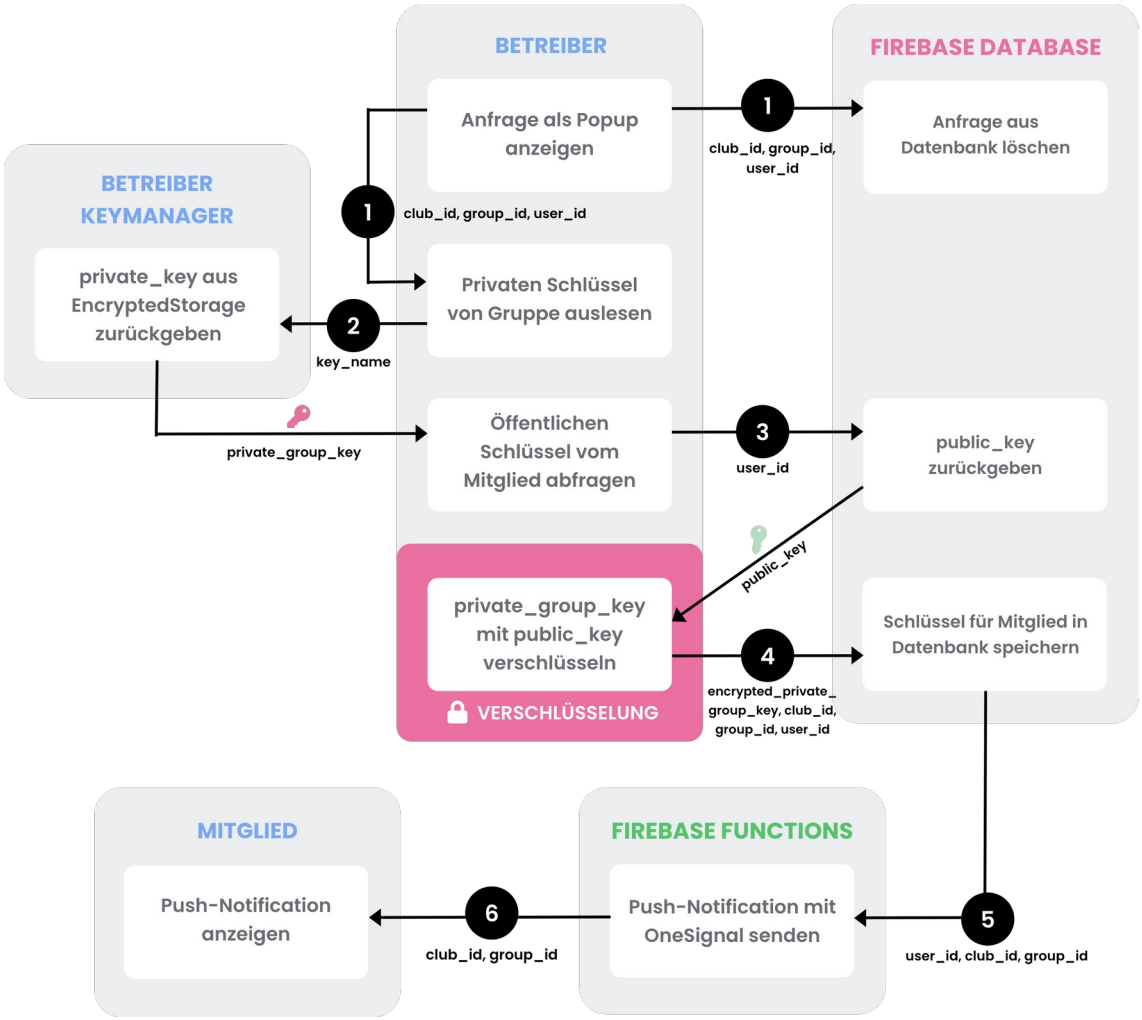


Abbildung 27: Austausch des privaten Gruppen-Schlüssels - Teil 2

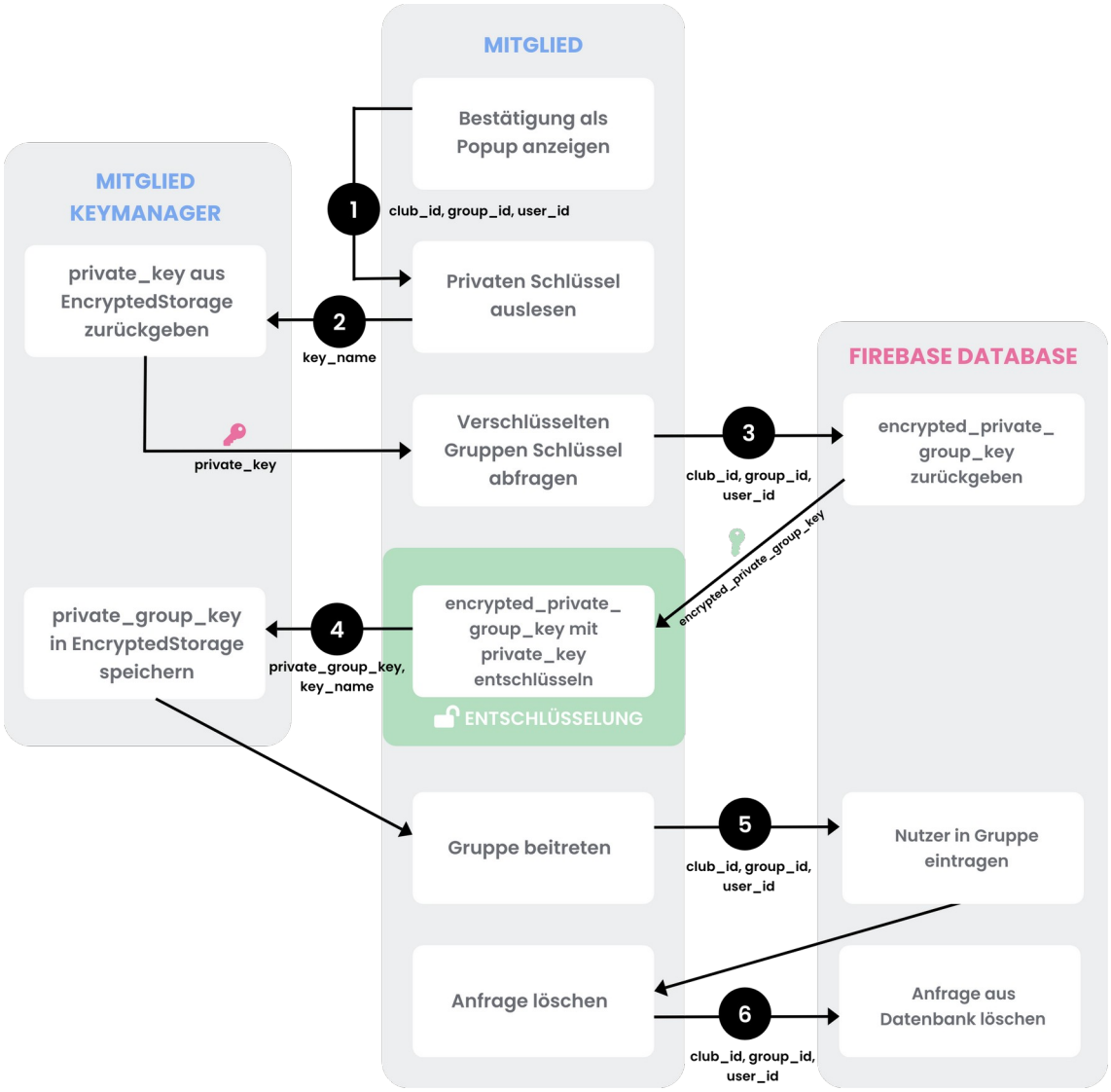


Abbildung 28: Austausch des privaten Gruppen-Schlüssels - Teil 3

Literaturverzeichnis

- HEISE ONLINE – TIPP: DARK MODE BEI WHATSAPP AM IPHONE AKTIVIEREN (07.07.2020): https://www.heise.de/ratgeber/Tipp-Dark-Mode-bei-WhatsApp-am-iPhone-aktivieren-4837862.html – Zugriff: 05.04.2021

- GOOGLE CLOUD – BUCKET-STANDORTE (02.04.2021): https://cloud.google.com/storage/docs/locations?hl=de#available-locations – Zugriff: 05.04.2021

- STATISA – DIE BELIEBTESTEN PROGRAMMIERSPRACHEN (...) IM APRIL (06.04.2021): https://de.statista.com/statistik/daten/studie/678732/umfrage/beliebteste-programmiersprachen-weltweit-laut-pypl-index/ - Zugriff: 06.04.2021

- GITHUB – REACT NATIVE ENCRYPTED STORAGE (März 2021): https://github.com/emeraldsanto/react-native-encrypted-storage – Zugriff: 06.04.2021

- HEISE ONLINE – WIE FACEBOOK MIT IHREN DATEN GELD VERDIENT (19.05.2012): https://www.heise.de/ct/ausgabe/2012-12-Wie-Facebook-mit-Ihren-Daten-Geld-verdient-2345376.html – Zugriff: 06.04.2021