卒業論文

Firebase を用いた
位置情報共有システム

提出者　　半田勝之

提出年月日　平成 31 年 1 月 25 日

指導教員　　金子邦彦 教授

Firebase を用いた
位置情報共有システム

情報工学科　半田勝之

研究概要

現在，スマートフォンの位置情報は，カーナビゲーションや地図などのアプリケーション，宅配の現在地表示サービスなど様々な目的で利用されている．本研究では，スマートフォンの位置情報やマーカー，チャットなどの情報をグローバルで共有し，コミュニケーションを取ることができる位置情報共有システムを試作した．Firebase と連携したアプリケーションの機能の動作と，Firebase 内のデータベースの動作を調べた．

目次

1. まえがき

現在，スマートフォンの GPS 機能は，地図アプリケーションや，カーナビアプリケーション，コミュニケーションツールなど様々な用途で活躍している．地図アプリケーションは，GPS に合わせて地図上に常に自分の位置や向きが表示される．目的の建物の位置に加えて，そこに到達するまでの経路を自分の位置と合わせて知りたい場合に便利である．また，自分の位置情報だけではなく，地図上で他人と位置情報を共有してコミュニケーションを取ることができる機能やアプリケーションも複数存在している．

位置情報を共有するアプリケーションや機能を利用する場合，電話番号，LINE の ID，Google アカウントなどの個人識別符号を介して共有することが考えられる．その場合，第三者がそれらの個人情報と位置情報を結びつけるのを防ぐために，公開する相手を限定する必要がある．安全ではあるが，個人識別符号を共有したくない場合は不便である．また，限定された相手との共有であるため，個人が地図上に追加した案内マーカーなどの情報を，不特定多数と共有したい場合にも向いていない．相手を限定せず，より多くの人と地図上で様々な情報を共有できれば，さらに便利となり社会的意義がある．

本論文では，個人識別符号を必要とせず，公開された地図上で位置情報や地図上の案内マーカーなどの共有が行える，位置情報共有システムについて説明する．本論文での位置情報共有システムは，ある特定の利用者グループにのみサービスを提供するシステムである．個人識別符号や認証などでデータアクセス制限を制御することは行わない．このとき，位置情報共有システムをいかに簡単に立ち上げることができるか，そして，個々の利用者グループの要望や特性を反映したアプリケーションをいかに簡単に制作できるかが課題になる．その課題の解決を目指して，位置情報共有システムの試作に取り組んだ．このとき，リアルタイムで情報を共有できるデータベースを構築する必要がある．しかし，そのようなデータベースを一から構築しようとすると，サーバの構築も必須となり，サーバの構築後の運用やアプリケーションとの連携の複雑な設定などが負担になるという課題がある．そこで，システムを簡単に開発することに焦点を当て，データベースに MBaaS である，Google 社の提供する「Firebase」を利用した．Firebase のデータベースが複数の利用者機器で簡単にリアルタイム共有でき，アプリケーションをいかに簡単に

制作できるかの確認に取り組んだ.

2. Firebase

位置情報共有システムのデータベースとして MBaaS である Firebase を利用した. MBaaS とは Mobile Backend as a Service の略であり, モバイルアプリケーション開発向けにバックエンドで動く機能を提供するクラウドサービスのことである. これにより, アプリケーションを開発する際にサーバの構築をすることなく, 通信を利用した機能を実装することができるという良さがある.

Firebase とは Google が提供している MBaaS である. Firebase の提供する主なサービスを以下に示す.

(1) Cloud Functions

専用サーバーの管理や拡張を行うことなく, カスタムのバックエンドコードでアプリを拡張する機能[3]がある.

(2) 認証

認証の機能である Firebase Auth では, メールやパスワード, Google や Facebook などのサードパーティのプロバイダ, 既存のアカウントシステムを直接使用するなど、複数の方法によって認証を行う機能[3]がある.

(3) ホスティング

ウェブアプリ専用に作成されたツールを使用して, 静的な Web ホスティングを行う機能[3]がある.

(4) クラウドストレージ

画像, 音声, 動画, またはその他のユーザーが生成したコンテンツを保存できるクラウドストレージの機能[3]がある.

(5)Realtime Database

ホスティングされた NoSQL データベースを使用して, リアルタイムでユーザーと端末の間での同期を行う機能[3]がある.

位置情報共有システムの試作にあたって, 位置情報のほかに名前や ID などの文字の情報を端末間で共有させるために Realtime Database を利用した. また, 写真を共有させるためにクラウドストレージを利用した. 詳しくは, 3 章で説明する.

3. 位置情報共有システム

位置情報共有システムは, iOS アプリケーションとデータベースから成る. データベースとアプリケーションの詳細をそれぞれ分けて後述する.

3.1 データベース

Firebase の Realtime Database を利用し, JSON 形式のリアルタイムデータベースを構築した. JSON 形式のデータベースは, すでに, 位置情報の共有以外にも, さまざまな目的で利用されており, 利用例は, Twitter に投稿されたテキストや位置情報などの複数の情報の管理(1)や, 地図(2)などがある.

データベースの構築手順は以下の通りである.

1. Google アカウントを作成.
2. Firebase にサインイン.
3. 「プロジェクトを追加」をクリックでプロジェクトを追加.
4. 「データベースを作成」をクリックでリアルタイムデータベースを作成.
5. テストモードで運用を開始.
6. 「iOS アプリに Firebase を追加」をクリックし, Google API を発行する. アプリのバンドル ID を入力し, 「GoogleService-Info.plist」ファイルをダウンロード.

以上は, Web ブラウザで操作した.

構築したリアルタイムデータベースを iOS アプリのプログラムであるプロジェクトファイルと結びつけるために, Mac 上でソースコードに Google API と GoogleServie-Info.plist ファイルを設定した.

詳しい手順は以下の通りである.

1. Xcode プロジェクトを作成
2. 「pod 'Firebase/Core'」のコマンドを「ターミナル」で実行し, Podfile にポッドを追加する.
3. 「pod install」でポッドをアプリのプロジェクトにインストールする.
4. プロジェクト内に GoogleServie-Info.plist を追加する.
5. プロジェクト内の UIApplicationDelegate に import Firebase と FirebaseApp.configure()の2つのプログラムを追加する.

以上で, Firebase との連携が完了する.

図 1 は，Web ブラウザ上のデータベースの管理画面であり，データベースの構造を示している．プロジェクト名を親とし，そこから目的ごとに複数の子に派生させ，データを格納した．「Chat」はユーザがチャットに投稿した内容を格納するノードである．「Latest」は最後に更新したユーザの位置情報を格納するノードである．「GPS」は「Latest」の緯度経度を格納するノードである．「Marker」はユーザが設置したマーカーの情報を格納するノードである．「userName」はユーザの ID や名前，コメントなどの情報を格納するノードである．各ノードのさらに深い階層については，機能ごとに後述（節 3.5）する．

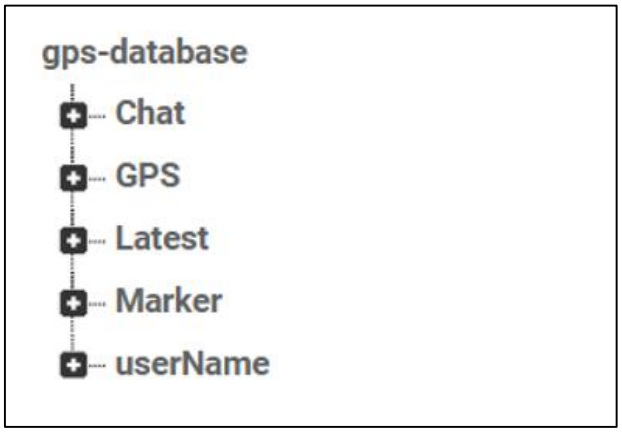


図 1　データベースの管理画面

3.2 iOS アプリケーション内での GPS の取得

ユーザの位置情報は，iOS 端末に搭載されている GPS センサーから 5 秒に 1 回の間隔で取得している．図 2 は，緯度・経度を取得した後，変数 latitude, longitude に代入しているコードである．

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {

        for location in locations {
            let latitude = location.coordinate.latitude
            let longitude = location.coordinate.longitude
        }
}
```

図 2　端末の位置情報を取得するコード

3.3 iOS アプリケーション内での Firebase の利用

アプリケーションとサーバの通信処理の実装は，図 3，図 4 のようなコードで行っている．図 3 は Firebase から情報をダウンロードするコードである．pick_json という変数にダウンロードした JSON データを代入している．図 4

は Firebase に情報をアップロードするコードである.「lastnode」はデータベースのパスを示す.updateChildValues()の中にアップロードする情報である, JSON の配列を指定している. コードの詳細は付録に示している.

```
self.ref.child(path).observeSingleEvent(of: .value, with: {(snapshot) in
        self.pick_json = JSON(snapshot.value as? [String : AnyObject] ?? [:])
})
```

図 3　データベースからの情報をダウンロードするコード

```
let lastnode = self.ref.child("Latest").child(AutoID_place.key)
lastnode.updateChildValues([
        "latitude": lat,
        "longitude": lon,
        "pressure": pre,
        "altitude": alt,
        "time": time
])
```

図 4　データベースに情報をアップロードするコード

3.4 アプリケーションの基本機能

位置情報共有システムを iOS アプリケーションとして試作した. アプリケーションで扱う情報は, データベースを介して他の端末と共有される.

3.4.1 ユーザ

次の機能を実現するにあたって, データベースの「Latest」と「userName」のノードを用いている.

- 自分自身の位置をマーカーと青い円で地図上に表示する機能.
- 他のユーザが最後に更新した位置をマーカーで地図上に表示する機能.
- マーカーの下にユーザ名を表示する機能.
- マーカーをタップすると編集可能な名前・コメントを表示する機能.

図 5 は「Latest」ノードの構造を示している. ユーザごとにノードを追加しており, その中に, 緯度・経度, タイムスタンプなどの情報が格納されている. ここから, 各端末が位置情報をダウンロード・アップロードし, 地図上のユーザ

を示すマーカーに反映する.

図 6 は「userName」ノードの構造を示している. ユーザごとにノードを追加しており, その中に, ユーザの名前やコメントの情報が格納されている. 端末がこのユーザの名前をデータベースからダウンロードし, 地図上のユーザの位置を示すマーカーの下に表示する名前に反映する.

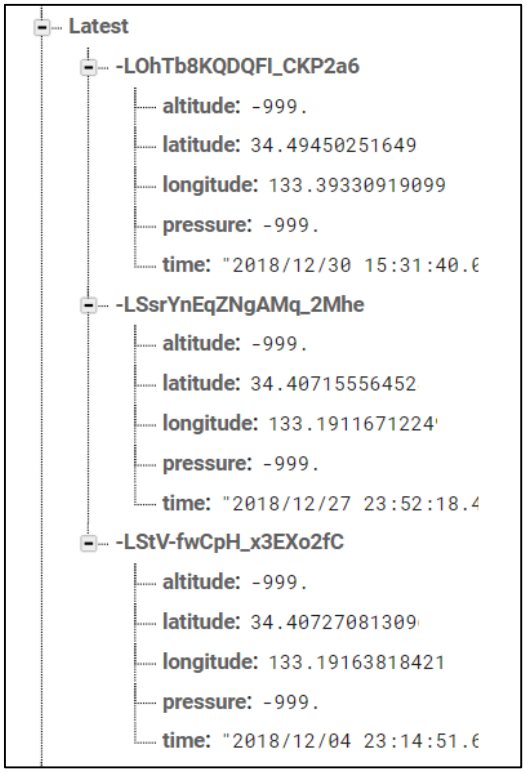


図 5 Latest の構造

図 6 userName の構造

3.4.2 マーカー

次の機能を実現するにあたって, データベースに「Marker」ノードを用いている. 画像ファイルは JSON 形式に変換せずに, Firebase のクラウドストレージで管理している.

- タップした場所にマーカーを設置する機能.
- 設置したマーカーをタップすると編集可能な名前・紹介文・写真を表示する機能.
- マーカーに追加された写真を保存する機能.
- 削除する機能.

図 7 は「Marker」ノードの構造を示している．設置されたマーカーごとにノードを追加しており，各ノードには，マーカーの色を RGB で表した「color」や，掲載している画像名を表す「image_name」のほかに緯度・経度や名前，コメントなどの情報を格納している．ここから，各端末が情報をダウンロード・アップロードし，地図上のマーカーに反映する．

図 8 は， Web ブラウザでクラウドストレージにアクセスしている画面である．マーカーに掲載する画像を蓄積・管理する．画像ファイル名が重複すると上書きされてしまうため，ファイル名は，マーカーの ID とした．マーカーの ID はマーカーを作成するときに自動的に付与され，他と重複することはない．

図 7 Marker の構造

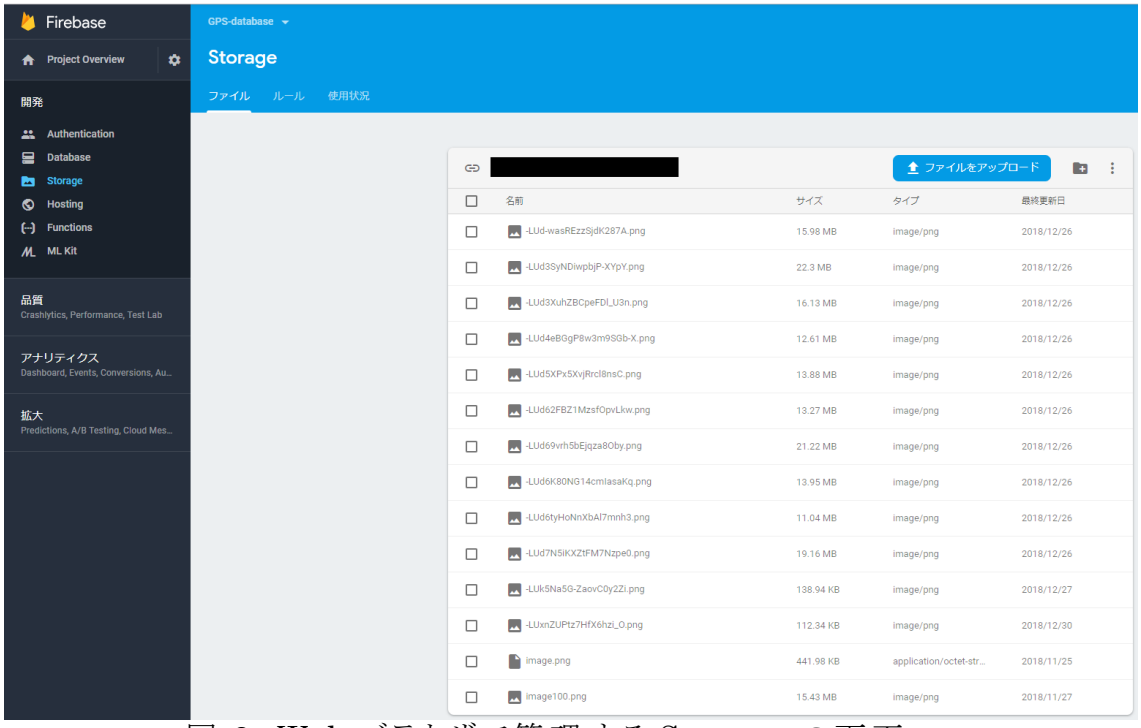


図 8　Web ブラウザで管理する Storage の画面

3.4.3 チャット

次の機能を実現するにあたって，データベースの「Chat」ノードを使用している．画像ファイルはクラウドストレージで管理している．

- グローバルなチャットルームにテキスト・写真を投稿する機能．
- チャット内の投稿された写真を保存する機能．

図 9 は「Chat」ノードの構造を示している．チャットに投稿されたメッセージごとにノードを追加している．その中で，投稿した画像名である「image_URL」やメッセージの内容である「message」，投稿したユーザの ID やタイムスタンプなどの情報を格納している．

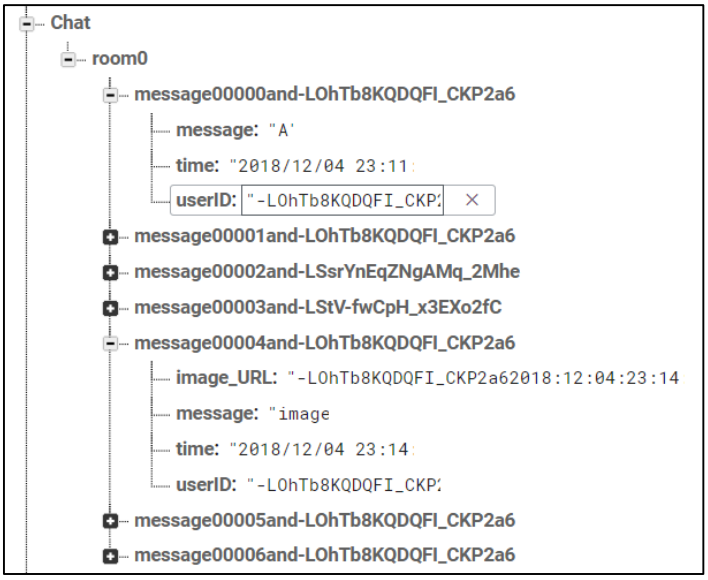


図 9　Chat の構造

これらの機能を実装したソースコードは付録の枠の部分に示した.

4. 評価実験

4.1 基本機能の実験概要

実験は 2 台の iOS 端末で行った. それぞれの端末で地図アプリを操作し, 節 3.4 で説明した機能が動作するかを, アプリケーションを実行している端末とデータベースを表示している Web ブラウザの画面を通して確認した.

データベースをテストモードで運用しており, 公開している状態になっている. そのため, 実験では, アプリケーション内のユーザ名やコメント, チャット・マーカーのテキストや写真にプライベートな情報は付与せず, 個人を特定できるような情報を位置情報と一緒にデータベースへアップロードしないように配慮した.

・ユーザ

①自分自身の位置をマーカーと青い円で地図上に表示する機能.

②他のユーザが最後に更新した位置をマーカーで地図上に表示する機能.

③マーカーの下にユーザ名を表示する機能.

④マーカーをタップすると編集可能な名前・コメントを表示する機能.

・チャット

⑤グローバルなチャットルームにテキストを投稿・表示する機能.

⑥グローバルなチャットルームに写真を投稿・表示する機能.

・マーカー

⑦タップした場所にマーカーを設置する機能.

⑧設置したマーカーをタップすると編集可能な名前・紹介文・写真を表示する機能.

⑨マーカーを削除する機能.

実験の手順は以下の通りである. 実験に使用した端末を端末 A, 端末 B とする.

(1)ホーム画面にある試作したアプリケーションのアイコンをタップし, 起動する.

(2)①から③は自動で行われるため, 操作せずに確認する.

(3)端末 A と端末 B を示すマーカーをタップして表示し, テキストを編集することで④を確認する.

(4)画面右端を左にスワイプすることで, チャットを開く. テキスト入力後, 投稿ボタンを押し, ⑤を確認する.

(5)写真選択ボタンを押し, 端末内の写真を選択後, 投稿ボタンを押すことで⑥を確認する.

(6)マーカー作成ボタンを押した後, 地図上の任意の場所をタップすることで⑦を確認する.

(7)作成されたマーカーをタップし, 名前・紹介文・写真が表示されるか確認する.

(8)名前・紹介文のテキストを変更した. 写真選択ボタンを押し, 写真選択後投稿ボタンを押し, ⑧を確認する.

(9)マーカーをタップした時に表示される削除ボタンを押し, ⑨を確認する.

また, データベースは各操作の度に, アプリケーションの情報が反映されているか Web ブラウザの画面を通して確認する.

4.2 基本機能の実験結果

2 つの iOS 端末の画面を通して，①から⑨までの機能が正常に動作していたのを確認した．また，位置情報やテキスト，写真などのデータがデータベースに反映されているか web ブラウザ上で確認できた．

図 10 は節 4.1 の手順の(3)を行った時の 2 端末の画面である．節 4.1 に示した，①②③④の機能が正常に動作していることが確認できた．

図 11 は節 4.1 の手順の(4)と(5)を行った時の 2 端末の画面である．節 4.1 に示した，⑤⑥の機能が正常に動作していることが確認できた．

図 12 は地節 4.1 の手順の(6)を行った時の 2 端末の画面である．節 4.1 に示した，⑦の機能が正常に動作していることが確認できた．

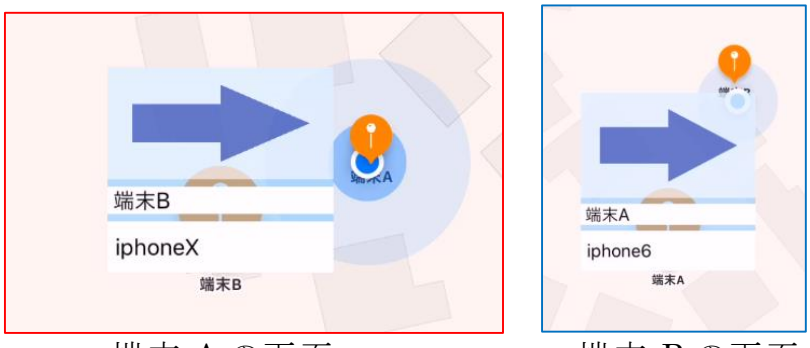
図 13 は節 4.1 の手順の(7)を行った時の 2 端末の画面である．

図 14 は節 4.1 の手順の(8)を行った時の 2 端末の画面である．図 13，図 14 から節 4.1 に示した，⑧の機能が正常に動作していることが確認できた．

図 15 は節 4.1 の手順の(9)を行った時の 2 端末の画面である．節 4.1 に示した，⑨の機能が正常に動作していることが確認できた．

図 16，17，18 は Firebase の Realtime Database を Web ブラウザ上で管理している画面である．各ノードにアプリケーションの機能によってアップロードされたデータが反映される様子が確認できた．

図 19，20 はクラウドストレージを Web ブラウザ上で管理している画面である．アプリケーションの機能によってアップロードされた写真が反映される様子が確認できた．

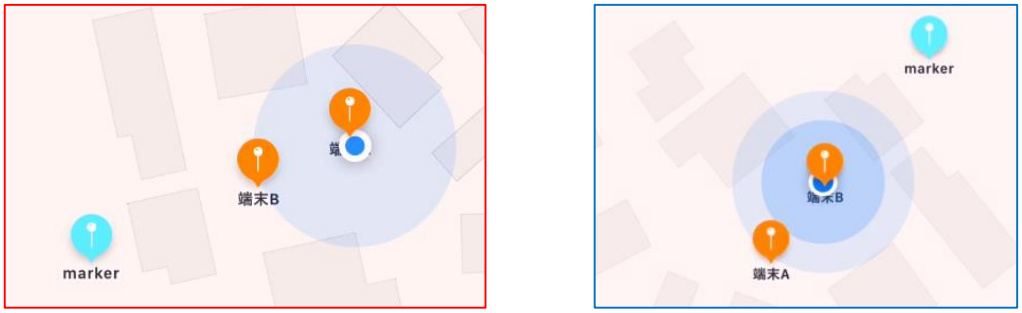


端末 A の画面　　端末 B の画面

図 10　ユーザを示すマーカーをタップした時の画面

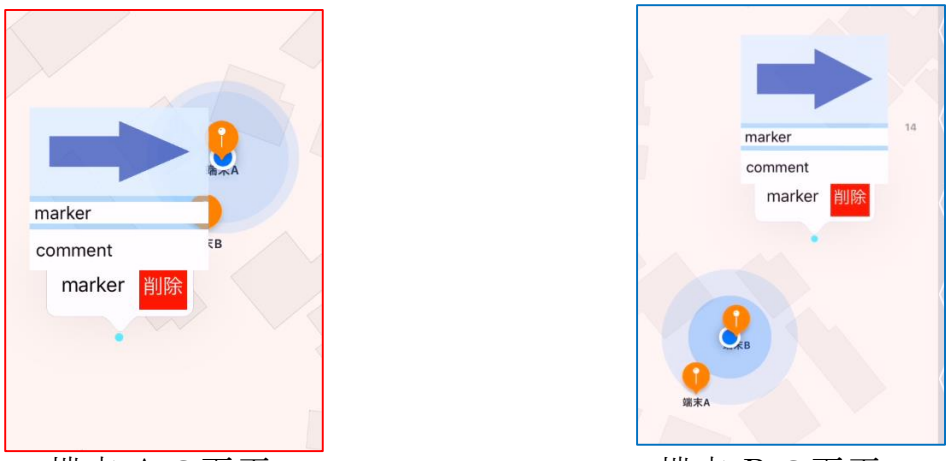
端末 A の画面　　端末 B の画面

図 11　チャット機能によるテキストと写真の投稿・共有



端末 A の画面　　端末 B の画面

図 12　地図をタップして，マーカーを設置した時の画面



端末 A の画面　　端末 B の画面

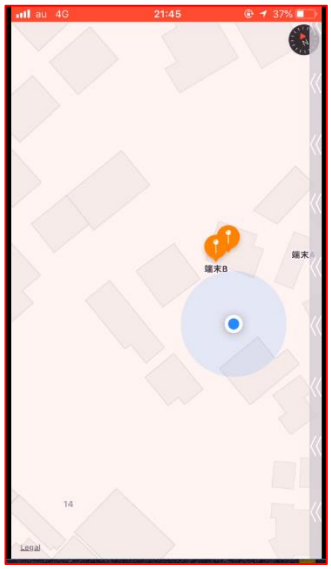
図 13　設置したマーカーをタップして，名前・テキストを表示している画面

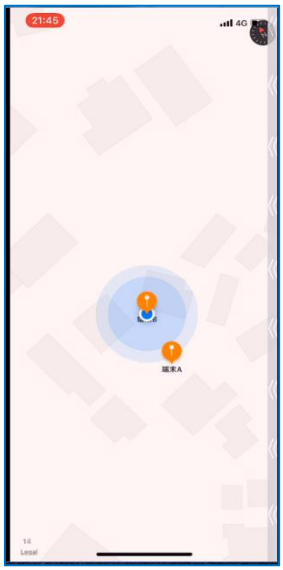
端末 A の画面

端末 B の画面

図 14　名前・テキスト・写真が変更されたマーカーを表示している画像

端末 A の画面





端末 B の画面

図 15　マーカーをタップすると表示される削除ボタンで削除した画面

図 16　Chat に投稿されたデータ

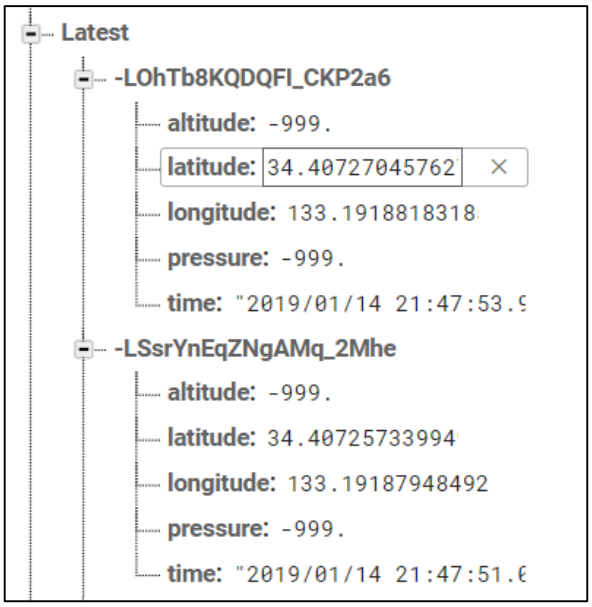


図 17　最新のユーザ情報

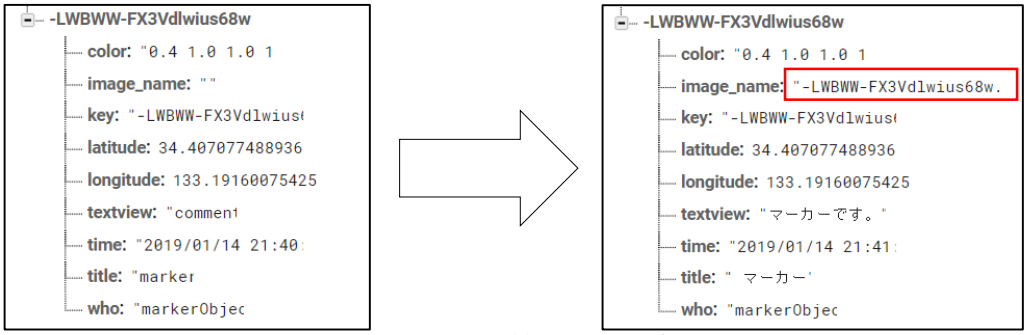


図 18　Marker 内のマーカー情報が更新される様子

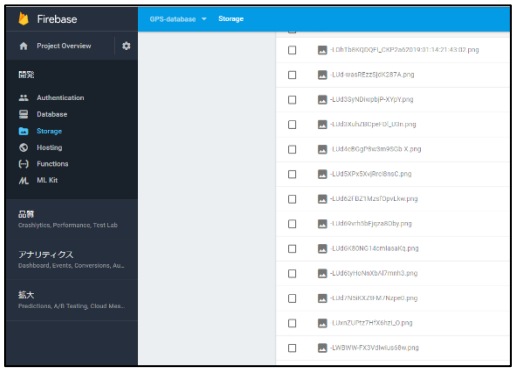


図 19　Storage の画面

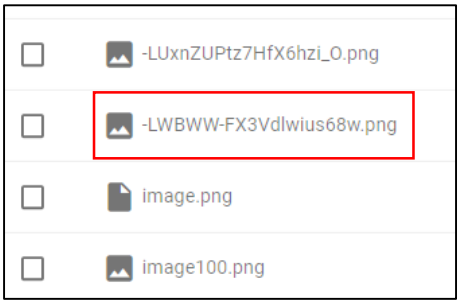


図 20　マーカーの画像名と一致

4.3 大学構内の探索，調査への応用（概要）

福山大学構内の探索，調査を 2 人で実施した．このとき，既存の地図にマーカーを設置し，要所の名前・写真・紹介文の情報をマーカーに掲載する作業を行った．使用した機材は，iPhone6 と iPhoneX の 2 端末である．1 つずつ端末を持ち，別々に行動し，アプリケーションの機能を使用して地図を完成させた．目的は，実際に使う場面で意図した通りに機能するのかを確認することである．

4.4 大学構内の探索，調査への応用（結果）

図 21 のように要所にマーカーを設置し，名前・紹介文・写真を掲載することができた．その情報を他の端末で確認することができた．また，事前に調査区域を分けていなかったが，お互いの位置や，追加したマーカーがリアルタイムで把握できたため，探索・調査に便利だった．このようにして，実際に地図を製作する際にも意図した通りに機能し，問題なく地図を製作することができた．

端末 A

端末 A

端末 B

端末 B

図 21　大学構内に追加した案内マーカーの画像

5. むすび

Firebase を利用したデータベースの構築が容易であり，特に課題なく iOS アプリと連携し，位置情報共有システムである，グローバルな地図アプリケーションの製作，機能の実装ができた．チャットや位置情報の共有によるコミュニケーションと案内マーカーを使ったグローバルな地図の製作が機能的には行える．

今後の課題は，通信量を減らすために，チャットやマーカーなどで扱う画像ファイルのサイズを小さくすることである．画像ファイルをアップロードする際に，圧縮することで実現できると思われる．

また，セキュリティ面は研究していないので，データベースが安全とは言い切れない．位置情報共有システムが製作できたとはいえ，セキュリティを考慮すると世界に向けてアプリケーションをリリースするレベルには達していないので今後の課題としたい．

謝辞

本研究の実施にあたり，卒業論文指導教員の情報工学科・金子邦彦教授にご指導を賜りました．金子邦彦研究室の飯塚氏，井上氏，田坂氏には，実験の協力，研究室や実験の場での議論等を通して，知識や示唆の提供をいただきました．ここに感謝の意を表します．本研究は科研費（15H05708）の助成を受けたものである．

参考文献

(1) 影澤秀明,廣井慧,奥矢淳,香取哲志,加藤朗,砂原秀樹:「Twitter を用いたセンシングシステムの提案と考察」，マルチメディア，分散，協調とモバイル（DICOMO2014）シンポジウム 平成 26 年 7 月．

(2) Mingzhao Li,Zhifeng Bao,Farhana Choudhury,Timos Sellis: “Supporting Large-scale Geographical Visualization in a Multi-granularity Way” WSDM '18 Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, pp.767-770, 2018.

(3) Google Firebase ホームページ,
https://firebase.google.com/?hl=ja
(4) Apple Developer Documentation
https://developer.apple.com/documentation
(5) 浮田 弥,山本 大介,高橋 直久:「逆進検知機能を有する案内粒度変更可能な音声経路案内システム」, DEIM Forum, 2017 I3-4

付録

位置情報共有システムの，地図アプリケーションのプログラムは

1. AppDelegate.swift
2. ViewController.swift
3. SideMenu.swift

の 3 つのファイルに分かれている．

付録では，「2.」を載せる．

1. 節 4.1 の機能①②③の処理を実装したソースコード

```
1403    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
1404
1405        for location in locations {
1406            let df = DateFormatter()
1407            df.dateFormat = "yyyy/MM/dd HH:mm:ss.SSSS"
1408            let timestamp: String = df.string(from: location.timestamp) //もともとyyyy/MM/dd HH:mm:ssだったのを変更
1409
1410            if location_span > 3 { //locationを呼ぶ頻度を減らす とりま5秒毎に設定
1411                if MakeMarker_ready == false && marker_selecting == false {//マーカー編集中に上書きされないように
1412                    if chat_update_now == false {
1413                        textSave(lat: location.coordinate.latitude, lon: location.coordinate.longitude, pre: pressure, alt: altitude, time: timestamp)
1414                    }
1415                }
1416                location_span = 0
1417            }else{
1418                location_span += 1
1419            }
1420
1421            if location_span_chat_marker > 8 {//10秒毎に設定
1422                if MakeMarker_ready == false && marker_selecting == false {//マーカー編集中に上書きされないように
1423                    if marker_update_now == false {
1424                        markerUpdate()
1425                    }
1426                    if chat_update_now == false {
1427                        ChatPartGet()
1428                    }
1429                }
1430                location_span_chat_marker = 0
1431            }else{
1432                location_span_chat_marker += 1
1433            }
1434
1435        }
1436    }
```

```
func textSave(lat: Double, lon: Double, pre: Double, alt: Double, time: String) { //json形式にする前にjsonの記述方式で記述された位置情報を保持しておく
    //Firebaseの処理----------------------------------------------------------------------
    //firebaseにデータをセットする++++++++++++++++++++++++++++++++++++++++++++++++++++++
    if map_ready_ON {
        let defaultplace = AutoID_place.child( String(GPS_count) )

        defaultplace.updateChildValues([
            "latitude": lat,
            "longitude": lon,
            "pressure": pre,
            "altitude": alt,
            "time": time
        ])

        let lastnode = self.ref.child("Latest").child(AutoID_place.key)
        lastnode.updateChildValues([
            "latitude": lat,
            "longitude": lon,
            "pressure": pre,
            "altitude": alt,
            "time": time
        ])

        //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++firebaseにデータをセットする
        //firebaseからデータを取得する++++++++++++++++++++++++++++++++++++++++++++++++++++++++


        let num = self.douki
        let getjson = self.pickup_JSON(path: "Latest", douki: num) //pick_up関数は下の方に記載

        let runLoop = RunLoop.current
        while keepAlive[num] &&
            runLoop.run(mode: RunLoopMode.defaultRunLoopMode, before: NSDate(timeIntervalSinceNow: 0.1) as Date) {
                // 0.1秒毎の処理なので、処理が止まらない
        }


        if getjson.string == "error" {
            print("Chat room nothing")
            return
        }else{

            var array: [String] = [] //第一層目のノード名のみを格納する配列
            for (key, val) in getjson.dictionaryValue { //ID(key)を抽出する JSON形式のデータKeyとValueで分ける
                array.append("\(key)")
            }
            array.sort(by: {$0 < $1} ) //Keyが辞書順になってないので昇順で並び替える

            for node in array{


                let latitude = getjson[node]["latitude"].double
                let longitude = getjson[node]["longitude"].double

                var exist:Bool = false
                var index:Int = 0
                for i in 0..<user_information.count { //すでにuser_informationに情報があるかどうかを検出する
                    if user_information[i].key == node {
                        exist = true
                        index = i
                    }
                }
                var textview_text:String? = nil
                var name = ""
                if exist == true {
                    name = user_information[index].name
                    textview_text = user_information[index].text
                }else{
                    name = ""
                    textview_text = "comment"
                }


                if latitude == nil || longitude == nil {
                    print("makePin error")
                }else{
                    if textview_text == nil {
                        textview_text = "comment"
                    }
                    makePin(key: node, lat: latitude!, long: longitude!, color: UIColor.orange, who: "user", title: name, text: textview_text!,upload: false, image: nil, image_name: "") //マーカーを置く関数
                }
            }
        }
    }
    GPS_count += 1
    userdefaults.set(GPS_count, forKey: "count")
}
```

```
func makePin(key: String, lat: Double, long: Double, color: UIColor, who: String, title: String, text: String, upload:Bool,image: UIImage?, image_name: String){ //makerを設置する　(注意：同じtitleのものは位置の更新)
    // ピンをつける
    var index:Int = 0
    pin_is = false
    var index_firebase:Int = 0

    for p in 0 ..< pin.count{// for文で回しているkeyと同じマーカーの名前がついたマーカーがあるか調べる
        if key == pin[p].key {
            pin_is = true
            index = p
        }
    }

    let center = CLLocationCoordinate2DMake(lat, long)

    let textview:UITextView? = UITextView(frame:  CGRect(x: 0, y: 0, width: 40, height: 40))
    //テキストを入れる
    textview?.text = text
    //フォントサイズ
    var fontsize: CGFloat = 15.0
    textview?.font = UIFont.systemFont(ofSize: fontsize)
    //テキストの量にサイズを合わせる
    textview?.sizeToFit()
    //編集不可にする
    textview?.isEditable = true
    // 背景色
    textview?.backgroundColor = UIColor(red: 119/255, green: 204/255, blue: 255/255, alpha: 1.0)

    if pin_is == false {// ピンがなかったら作成

        pin.append(MKPointAnnotation())
        pin[pin.count - 1].coordinate = center
        pin[pin.count - 1].key = key
      //本来MKAnnotationには markerColorという変数は無い。extensionでUIColorを保持するために新たな変数として追加した。
        pin[pin.count - 1].markerColor = color
      //本来MKAnnotationには whoという変数は無い。extensionでStringを保持するために新たな変数として追加した。
        pin[pin.count - 1].who = who
      //本来MKAnnotationには imageという変数は無い。extensionでUIImageを保持するために新たな変数として追加した。
        if image != nil {
            pin[pin.count - 1].image = image
        }
        if image_name != "" {
            pin[pin.count - 1].image_name = image_name
        }

        if title == "" {
            if who == "markerObject"{
                pin[pin.count - 1].title = "marker"
            }else if who == "user"{
                pin[pin.count - 1].title = "user"
            }else{
                pin[pin.count - 1].title = "else"
            }
        }else{
            pin[pin.count - 1].title = title
        }

        pin[pin.count - 1].textview = textview
        pin[pin.count - 1].name = title

        self.mapView.addAnnotation(pin[pin.count - 1])

        index_firebase = pin.count - 1

        if who == "marker_circle" {
            //circle add----------------------------------------------------
            pin[pin.count - 1].circle = MKCircle(center: center, radius: 100)
            self.mapView.add(pin[pin.count - 1].circle!)
            //----------------------------------------------------circle add
        }
    }else{// ピンがあれば位置を更新
        pin[index].coordinate = CLLocationCoordinate2DMake(lat, long)
        pin[index].textview = textview
        pin[index].name = title
        pin[index].title = title
        pin[index].markerColor = color
        pin[index].who = who
        if image != nil {
            pin[index].image = image
        }
        if image_name != "" {
            pin[index].image_name = image_name
        }
```

```
1758
1759            print("ピンを更新")
1760
1761            if who == "marker_circle" {
1762                //circle update(remove -> add)-----------------------------------
1763                if pin[index].circle != nil {
1764                    /* MakeMarker_ON == true, MakeCircle_ON == false で一旦マーカーを置いた後に
1765                       MakeMarker_ON == true, MakeCircle_ON == true で上書きした場合にエラーが起きないようにする*/
1766                    self.mapView.remove(pin[index].circle!)
1767                }
1768                pin[index].circle = MKCircle(center: center, radius: 100)
1769                self.mapView.add(pin[index].circle!)
1770                //-------------------------------------circle update(remove -> add)
1771            }
1772
1773            index_firebase = index
1774        }
1775
1776        //firevaseに記録-------------------------------------------------
1777        if upload == true {
1778            //who == user の場合は既にfirebaseのchild(GPS)内にあるため追加しない
1779            if who == "markerObject" || who == "marker_circle"{
1780                let defaultplace = self.ref.child("Marker").child( String(key) )
1781
1782                let formatter: DateFormatter = DateFormatter()
1783                formatter.calendar = Calendar(identifier: Calendar.Identifier.gregorian)
1784                formatter.dateFormat = "yyyy/MM/dd HH:mm:ss"
1785                let now = Date()
1786                let Now: String = formatter.string(from: now)
1787
1788                //marker_Color (UIColor) をRGB変換している
1789                var R:CGFloat = 0.0
1790                var G:CGFloat = 0.0
1791                var B:CGFloat = 0.0
1792                var alpha:CGFloat = 0.0
1793
1794                let components = pin[index_firebase].markerColor?.cgColor.components!
1795                R = components![0]
1796                G = components![1]
1797                B = components![2]
1798                alpha = components![3]
1799                let RGB:String = String("\(R)"+" "+"\(G)"+" "+"\(B)"+" "+"\(alpha)")
1800                if pin[index_firebase].image_name == nil {
1801                   pin[index_firebase].image_name = ""
1802                }
1803
1804                defaultplace.updateChildValues([
1805                    "key": pin[index_firebase].key!,
1806                    "title": pin[index_firebase].title!,
1807                    "textview": pin[index_firebase].textview?.text,
1808                    "latitude": pin[index_firebase].coordinate.latitude,
1809                    "longitude": pin[index_firebase].coordinate.longitude,
1810                    "color": RGB,
1811                    "who": pin[index_firebase].who,
1812                    "time": Now,
1813                    "image_name": pin[index_firebase].image_name
1814                ])
1815            }
1816        }
1817        //-------------------------------------------------firevaseに記録
1818
1819    }
```

2. 節 4.1 の機能④⑧の処理を実装したソースコード

```
func mapView(_ mapView: MKMapView, didSelect view: MKAnnotationView) {
    print("func mapView(_ mapView: MKMapView, didSelect view: MKAnnotationView) { Did Select!!!!!!")
    if view.annotation is MKUserLocation {
        return
    }

    if marker_selecting == true { //マーカーを選択中にさらに他のマーカーを選択した場合にdidDeselectが呼ばれないことを防ぐ。
        return
    }

    putAnnotationKey(sender: (view.annotation?.key)!) //選択しているpinのkeyを保存しておく

    var image_height: CGFloat = 0.0
    let ImageView = UIImageView()

    if view.annotation?.image != nil{ //image画像があるならば
        image_height = mapView.frame.size.height / 3
        ImageView.image = view.annotation?.image
        ImageView.tag = 3
    }else{ //ないなら画像追加用のボタン(画像)を設置
        image_height = mapView.frame.size.height / 9
        let image_buf = UIImage(named:"enter.png")!
        ImageView.image = image_buf
        ImageView.alpha = 0.6
        ImageView.tag = -3
    }
    // サイズ設定
    let width: CGFloat = mapView.frame.size.width * 4 / 10
    ImageView.frame.size.width = width
    ImageView.frame.size.height = image_height
    // 位置設定
    ImageView.frame.origin.x = 0
    ImageView.frame.origin.y = 0

    // UIImageView がタップできるようにイベント追加
    ImageView.isUserInteractionEnabled = true
    //タッチした時のデリゲートを設定
    ImageView.addGestureRecognizer(UITapGestureRecognizer(target: self, action: #selector(self.onClickImageView(sender:))))


    view.annotation?.hukidasi = UIView(frame: CGRect(x: 0, y: -image_height + 5, width: mapView.frame.size.width * 4 / 10, height: mapView.frame.size.height / 2 ))
    view.annotation?.hukidasi?.backgroundColor = UIColor(red: 119/255, green: 204/255, blue: 255/255, alpha: 0.5)//UIColor.white

    let textField:UITextField? = UITextField(frame: CGRect(x: 0, y: image_height + 5, width: mapView.frame.size.width * 4 / 10, height: mapView.frame.size.height / 2 ))

    //フォントサイズ
    let fontsize: CGFloat = 15.0
    textField?.font = UIFont.systemFont(ofSize: fontsize)
    //編集可にする
    textField?.isEnabled = true
    // 背景色
    //textField?.backgroundColor = UIColor(red: 119/255, green: 204/255, blue: 255/255, alpha: 0.5)
    textField?.backgroundColor = UIColor.white
    if view.annotation?.name == nil {
        textField?.text = " name"
    }else{
        if (view.annotation?.name)!.prefix(1) != " " {
            textField?.text = " " + (view.annotation?.name)!
        }else{
            textField?.text = (view.annotation?.name)!
        }
    }
    //テキストの量にサイズを合わせる
    textField?.sizeToFit()
    textField?.frame.size.width = mapView.frame.size.width * 4 / 10
    textField?.tag = 0
    textField?.delegate = self

    let textview:UITextView? = UITextView(frame:  CGRect(x: 0, y: image_height + 5 + (textField?.frame.size.height)! + 5, width: mapView.frame.size.width * 4 / 10, height: mapView.frame.size.height / 2 ))
    //テキストを入れる
    if view.annotation?.textview == nil {// アノテーションビューにtextviewを設置
        textview?.text = "view?.annotation?.textview == nil"
    }else{
        textview?.text = view.annotation?.textview?.text!
    }
    //フォントサイズ
    textview?.font = UIFont.systemFont(ofSize: fontsize)
    //テキストの量にサイズを合わせる
    textview?.sizeToFit()
    textview?.frame.size.width = mapView.frame.size.width * 4 / 10
    //編集可にする
    textview?.isEditable = true
    // 背景色
    //textview?.backgroundColor = UIColor(red: 119/255, green: 204/255, blue: 255/255, alpha: 0.5)
    textview?.backgroundColor = UIColor.white
    textview?.tag = 1
    textview?.delegate = self


    view.annotation?.hukidasi?.addSubview(ImageView)
    view.annotation?.hukidasi?.addSubview(textField!)
    view.annotation?.hukidasi?.addSubview(textview!)
    view.annotation?.hukidasi?.sizeToFit()

    view.annotation?.hukidasi?.frame.size.width = CGFloat(mapView.frame.size.width * 4 / 10)
    view.annotation?.hukidasi?.frame.size.height = CGFloat( (textField?.frame.size.height)! + 5.0 + (textview?.frame.size.height)! ) + image_height + 5.0
    view.annotation?.hukidasi?.frame.origin.x =  -(view.annotation?.hukidasi?.frame.width)! / 2 + 10
    view.annotation?.hukidasi?.frame.origin.y = -((view.annotation?.hukidasi?.frame.height)! + 30 ) //koko


    let coordinate =  CLLocationCoordinate2D(latitude: (view.annotation?.coordinate.latitude)!,longitude: (view.annotation?.coordinate.longitude)!)
    var point:CGPoint = mapView.convert(coordinate, toPointTo: mapView)
    point.x -= CGFloat(mapView.frame.size.width * 2 / 10)
    var sa:CGFloat = 0.0
    if view.annotation?.who != nil {
        if view.annotation?.who! == "user"{
            sa = CGFloat( (textField?.frame.size.height)! + 5.0 + (textview?.frame.size.height)! + image_height + 5.0)
            point.y -= sa
        }else{
            sa = CGFloat(60 + CGFloat( (textField?.frame.size.height)! + 5.0 + (textview?.frame.size.height)! + image_height + 5.0))
            point.y -= sa
        }
    }
    view.annotation?.hukidasi?.frame.origin = point
    view.annotation?.hukidasi?.tag = -99 // didchange のマップが動いた時に追従させる時に subviewsからviewを判別して取り出すためにtagを設定
    view.annotation?.hukidasi_and_maker_point_sa = CGPoint(x: CGFloat(-(mapView.frame.size.width * 2 / 10)), y: -sa)



    mapView.addSubview((view.annotation?.hukidasi)!)

    marker_selecting = true
}
```

3. 節 4.1 の機能⑤⑥の処理を実装したソースコード

```
func chatupdate(mode: String){
    if chat_update_now == true {
        return
    }
    chat_update_now = true

    let num = self.douki
    let getjson2 = self.pickup_JSON(path: "Chat/room0", douki: num) //pick_up関数は下の方に記載

    let runLoop = RunLoop.current
    while keepAlive[num] &&
        runLoop.run(mode: RunLoopMode.defaultRunLoopMode, before: NSDate(timeIntervalSinceNow: 0.1) as Date) {
            // 0.1秒毎の処理なので、処理が止まらない
    }


    if getjson2.string == "error" {
        print("Chat room nothing")
        chat_update_now = false
        return
    }else{

        var array: [String] = [] //第一層目のノード名のみを格納する配列
        for (Key, val) in getjson2.dictionaryValue { //ID(key)を抽出する JSON形式のデータKeyとValueで分ける
            array.append("\(Key)")
        }
        array.sort(by: {$0 < $1} ) //Keyが辞書順になってないので昇順で並び替える


        if mode == "part" { //一部だけの時は既にあるメッセージは省く
            if self.sideView.textview_count + self.sideView.imageview_count != 0 { // 画面上に既にチャットメッセージがある場合,表示されている数だけ省く
                if self.sideView.textview_count + self.sideView.imageview_count == array.count{
                    print("Chat Same -> return")
                    chat_update_now = false
                    return
                }
                //配列内の指定した範囲を削除する
                print("self.sideView.textview_count:",self.sideView.textview_count)
                print("self.sideView.imageview_count:",self.sideView.imageview_count)
                print("textview_count + imageview_count:",self.sideView.textview_count + self.sideView.imageview_count)
                print("array.count:",array.count)

                if self.sideView.textview_count + self.sideView.imageview_count > array.count{
                    print("array.count < chat.count  ---> return")
                    chat_update_now = false
                    return
                }
                array.removeSubrange(0..<self.sideView.textview_count + self.sideView.imageview_count)
            }
        }


        for node in array{
            chat_update_now = true

            print("getjson2+++++++++++++++++++++++++++++++++++++++++")
            if getjson2.string != "error" {
                let name = getjson2[node]["userID"].string
                let text = getjson2[node]["message"].string
                let time = getjson2[node]["time"].string
                let image_URL = getjson2[node]["image_URL"].string
                //var image_size:CGFloat = 50
                var Image:UIImage!
                print("-----------------------------------")
                print("image_URL",image_URL)


                if image_URL != nil { //image_URLがある場合、messageではなくimageをセット
                    print("2-----------------------------------")
                    //Image
                    if name == nil || text == nil || time == nil {
                        print("Chat all get error")
                        if node == array[array.count - 1] { //最後のfor文だった場合
                            self.chat_update_now = false
                        }

                    }else{

                        print("Chat all of one get!!")
                        var which: String = "left"
                        if name! == self.AutoID_place.key {
                            which = "right"
                        }
                        //firebaseStorageから画像をダウンロードして、Imageに代入
                        let imageRef = self.ref_sto.child(image_URL!)
                        print("Chat all of one get!!22222")

                        self.imageView.append(UIImageView())
                        let index = self.imageView.count - 1
                        // サイズ設定
                        self.imageView[index]?.frame.size.width = UIScreen.main.bounds.height / 2
                        self.imageView[index]?.frame.size.height = UIScreen.main.bounds.height / 2
```

```
                    print("Chat all of one get!!33333")
                    //print("imageRef: ",imageRef)


                    self.keepAlive_download.append(true)
                    let num:Int = self.keepAlive_download.count - 1
                  //firebase Storage から画像をダウンロードする
                    //この中でチャットへのメッセージの追加も行っている
                    let downloadTask = self.downloadimage(index: num, imageref: imageRef, name: name!, time: time!, which: which)


                    print("4444444444444444444444444")
                    let runLoop2 = RunLoop.current
                    while self.keepAlive_download[num] &&
                        runLoop2.run(mode: RunLoopMode.defaultRunLoopMode, before: NSDate(timeIntervalSinceNow: 0.1) as Date) {
                            // 0.1秒毎の処理なので、処理が止まらない
                    }
                    self.keepAlive_download[num] = true

                    //downloadが成功すれば実行
                    downloadTask.observe(.success){ snapshot -> Void in
                        print("count: ",self.down_count)
                        let count = self.down_count
                        print("self.imageView?.image",self.imageView[count]?.image)
                        if self.imageView[count]?.image != nil { // downloadした画像がnilじゃなかった場合
                            Image = self.imageView[count]?.image
                            self.sideView.chatSet(name: name!, time: time!, image: Image!, Which: which)

                        }else{// downloadした画像がnilだった場合
                            //downloadが失敗すればもともとある画像に置き換えて実行する
                            print("SSSSSSSSSSSSSSSSSSSSSSSSS")
                            if Image == nil{
                                let gazou = UIImage(named:"failed.png")!
                                Image = gazou
                            }
                            self.sideView.chatSet(name: name!, time: time!, image: Image!, Which: which)
                        }
                        if node == array[array.count - 1] { //最後のfor文だった場合
                            self.chat_update_now = false
                        }
                        self.down_count += 1
                    }

                    //downloadが失敗すればもともとある画像に置き換えて実行する
                    downloadTask.observe(.failure){ snapshot -> Void in
                        // 「画像を取得できませんでした」の画像に置き換える
                        print("sippaiiiiiiiiiiiiiiiiiiiiiiii")

                        let Image = UIImage(named:"failed.png")!
                        self.sideView.chatSet(name: name!, time: time!, image: Image, Which: which)
                        if node == array[array.count - 1] { //最後のfor文だった場合
                            self.chat_update_now = false
                        }
                        self.down_count += 1
                    }
                }

            }else{//image_URLが無い場合（textmessageの場合） imageではなくmessageをセット
                if node == array[array.count - 1] { //最後のfor文だった場合
                    chat_update_now = false

                    DispatchQueue.main.asyncAfter(deadline: .now() + 0.2) {
                        if name == nil || text == nil || time == nil {
                            print("Chat all get error")
                        }else{
                            print("Chat all of one get!!")
                            var which: String = "left"
                            if name! == self.AutoID_place.key {
                                which = "right"
                            }
                            self.sideView.chatSet(name: name!, time: time!, Text: text!, Which: which)
                            self.chat_update_now = false
                        }
                    }
                }else{
                    DispatchQueue.main.asyncAfter(deadline: .now() + 0.2) {
                        if name == nil || text == nil || time == nil {
                            print("Chat all get error")
                        }else{
                            print("Chat all of one get!!")
                            var which: String = "left"
                            if name! == self.AutoID_place.key {
                                which = "right"
                            }
                            self.sideView.chatSet(name: name!, time: time!, Text: text!, Which: which)
                        }
                    }
                }
            }
        }
    }
}
```

4. 節 4.1 の機能⑦の処理を実装したソースコード

```
@objc func mapViewDidTap(sender: UITapGestureRecognizer) {
    var who:String = ""
    if MakeCircle_ready == true && MakeMarker_ready == true{
        who = "marker_circle"
    }else{
        who = "markerObject"
    }

    if MakeMarker_ready == true {
        if sender.state == UIGestureRecognizerState.ended {
            let tapPoint = sender.location(in: view)
            let center = mapView.convert(tapPoint, toCoordinateFrom: mapView)


            makePin(key: Marker_ID, lat: center.latitude, long: center.longitude, color: UIColor(red: 0.4, green: 1.0, blue: 1.0, alpha:1.0), who: who,title: "", text:"comment",upload: true, image: nil, image_name: "")
        }
    }
}
```

5. 節 4.1 の機能⑨の処理を実装したソースコード

```
func removesPin(view: MKAnnotationView, mapView: MKMapView){
    view.annotation?.remove_hukidasi_subview() //hukidasilに含まれるsubviewとhukidasiを削除しておく
    if view.annotation!.who! == "marker_circle" { //円付きマーカーの場合、円を削除しておく
        self.mapView.remove(view.annotation!.circle!)
    }
    if view.annotation!.who! == "markerObject" || view.annotation!.who! == "marker_circle" { //マーカーだけ削除　ユーザーは削除しない
        //firebaseから削除-------------------------------------
        let defaultplace = self.ref.child("Marker").child( String(view.annotation!.key!) )
        defaultplace.removeValue()
        print(String(view.annotation!.key!))
        //---------------------------------------firebaseから削除
        // Firebase Storage の画像を削除する-----------------------------------
        if view.annotation?.image != nil {
            let imageRef = ref_sto.child( String((view.annotation?.key)!) + ".png")
            imageRef.delete { error in
                if error != nil {
                    print("Uh-oh, an error occurred!")
                } else {
                    print("Firebase Storage image ----- delete success!!!!!")
                }
            }
        }
        //-------------------------------------Firebase Storage の画像を削除する
        let key = view.annotation?.key //削除する前にkeyを保存しておく

        self.mapView.removeAnnotation(view.annotation!)

        if key != nil {
            for p in 0 ..< pin.count{// for文で回しているkeyと同じマーカーの名前がついたマーカーがあるか調べる
                if key! == pin[p].key! {
                    pin.remove(at: p)
                    break
                }
            }
        }
    }
}
```