

# ODA Data TMF-681 Communication knowledge

v0.0.1

## Get Start

Git : <https://github.com/corp-ais/next-myais2-oda-data-tmf681-communication>

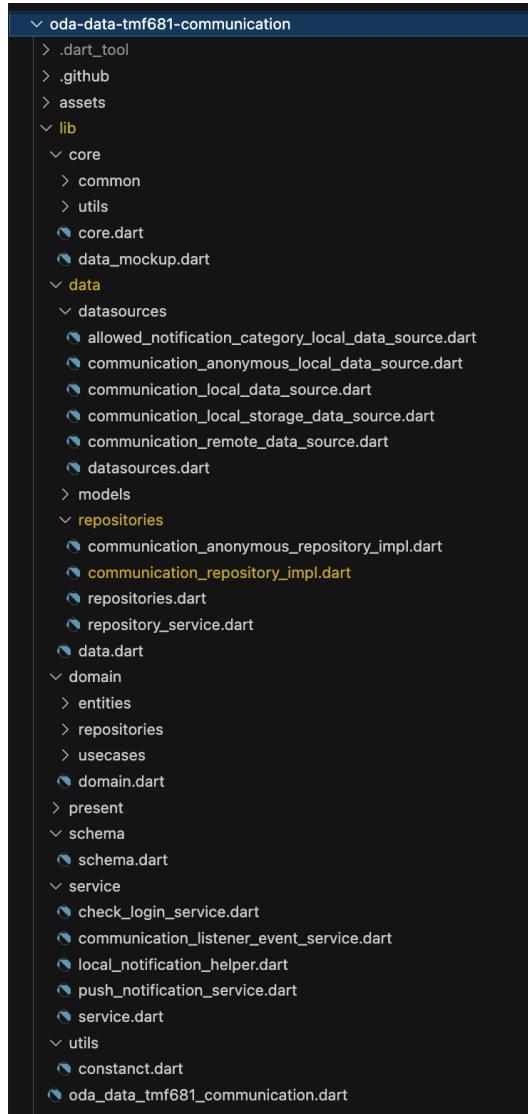
Technical Design :

[https://mimotech-my.sharepoint.com/:f/g/personal/ninalawa\\_ais\\_co\\_th/Eu3PPOGGX8hGqzm6nXWwkEEBSqyTKkpEIPRVIPbaLrfXYw?e=IJfjDX](https://mimotech-my.sharepoint.com/:f/g/personal/ninalawa_ais_co_th/Eu3PPOGGX8hGqzm6nXWwkEEBSqyTKkpEIPRVIPbaLrfXYw?e=IJfjDX)

## ODA Data TMF-681 Communication Function

1. In App Function
2. Push Notification
3. Notification Service

# Structure



## Data

### DataSource

datasource เป็น path สำหรับ service ในการเข้ามต่อและจัดการ raw data ในส่วนต่างๆ

allowed\_notification\_category\_local\_data\_source

ใช้สำหรับจัดการข้อมูลการ setting notification บน realm

communication\_anonymous\_local\_data\_source

ใช้สำหรับจัดการข้อมูล CommunicationMessageAnonymous  
ที่ใช้สำหรับการ Refferent message และ user action บน realm

communication\_local\_data\_source

ใช้สำหรับจัดการข้อมูล communicationMessage และ  
communicationMessageMaster บน realm

communication\_remote\_data\_source

ใช้สำหรับจัดการข้อมูลที่จำเป็นจะต้องเชื่อมต่อ service ภายนอก ทั้ง rest และ socket

## Repository

ใช้สำหรับทำ flow process และ logic การเรียกใช้ servive / datasouce เป็นตัวหลักในการ  
จัดการ process ของ usecase

communication\_anonymous\_repository\_impl

ใช้สำหรับทำ flow process และ logic ขณะ user 'ไม่ได้ login'

communication\_repository\_impl

ใช้สำหรับทำ flow process และ logic ขณะ user 'ได้ login' และ

repository\_service

ใช้สำหรับทำ service process และ logic กลางให้ repository อื่นเรียกใช้งาน

## Domain

### Entities

entities เป็น data model สำหรับ persentation ใช้จัดการและแสดงผล

### Repository

repository เป็น abstract class สำหรับ กำหนด คุณสมบัติหลัก ของ repository implement

## Usecase

Usecase เป็น interface สำหรับให้ presentation, oda เรียกใช้งาน

### view\_communication\_messages\_usecase

ใช้ interface และ logic เป็นต้นขณะ สำหรับให้ presentation, oda เรียกใช้งาน

## Service

Service ใช้สำหรับทำ flow process และ logic การเรียกใช้ service อื่นๆ, repository, datasouce เป็นตัวหลักในการจัดการ process ที่ไม่ได้เรียกผ่าน usecase

### check\_login\_service

ใช้สำหรับจัดการข้อมูล user login/ store data / asset มีความสามารถล้าย usecase ของ tmf 637 party

### communication\_listener\_event\_service

ใช้สำหรับจัดการ core event, stream event ที่เกิดขึ้นภายใน app ที่ communication จะเป็นต้องจัดการร่วมใน event นั้นๆ

### local\_notification\_helper

ใช้สำหรับจัดการ การแสดง notification, การกด notification ที่เกิดขึ้นขณะ app อยู่ในสถานะ foreground

### push\_notification\_service

ใช้สำหรับจัดการ notification service อื่นๆ ทั้งหมด, จัดการ fcm token, จัดการการ sub/un-sub topic ของ device, การ initail service สำหรับ notification

## Util

util เป็นเครื่องมือกลางในการส่วนจัดการ หรือใช้งานในส่วนอื่นๆ

### constanct

ใช้ในการประกาศค่ากลาง จัดเก็บค่ากลางสำหรับใช้งานในส่วนอื่นๆ

## In App Function

### การดึง message inbox ทั้งหมด

การดึง message inbox ทั้งหมด จะทำการเรียก ผ่าน usecase getAllNotificationInbox โดยจะต้องส่ง parameter List<UserAssetModel> assets เพื่อใช้ในการ map tag ของ asset ที่ได้มีการส่งเข้ามา และเมื่อต้องการ filter message inbox เฉพาะ asset ที่ต้องการ สามารถส่ง parameter filterAsset เข้ามาเพิ่มเติมได้

โดยในส่วนของ usecase getAllNotificationInbox จะทำการ check login ก่อนเพื่อเลือกใช้งาน repository ตาม role login

```
Future<Either<Failure, Map<String, NotificationEntity>>>
    getAllNotificationInbox(List<UserAssetModel> assets,
                           {UserAssetModel? filterAsset}) async {
    var islogin = await isLogin();
    if (islogin) {
        return repository.getAllNotification(
            assets,
            filterAsset: filterAsset,
        );
    } else {
        return anonymousRepositoryImpl.getAllNotification();
    }
}
```

### กรณีที่ทำการ login และ

จะใช้งาน CommunicationRepositoryImpl และส่ง parameter ตามที่ได้รับมา CommunicationRepositoryImpl เมื่อเริ่มทำงานจะทำการสร้าง map object สำหรับเก็บ result ของ badge และ notificationList ขึ้นมา โดยจะใส่ object ตาม category ที่แสดงบนหน้าแอปโดยจะทำการรีเมท all ก่อนแล้วทำการเพิ่ม category อื่นๆ ผ่าน loop โดยใช้ค่าจาก default\_category\_notification ที่อยู่ใน constant เมื่อกำหนด map object สำหรับเก็บ result เมื่อต้นแล้ว จะทำการ query allRefTempMessages จาก CommunicationMessageAnonymousRepositoryImpl ที่เก็บ message refferent เอาเพื่อใช้ในการเปลี่ยนเที่ยบ message ต่อไป

```
91     Map<String, int> baged = {"all": 0};
92     Map<String, List<BehaviorSubject<NotificationModel>>> notificationList = {
93         "all": []
94     };
95
96     for (String logicalCategory in default_category_notification) {
97         baged.addAll({logicalCategory: 0});
98         notificationList.addAll({logicalCategory: []});
99     }
100
101    List<CommunicationMessageAnonymous> allRefTempMessages =
102        await communicationMessageAnonymousRepositoryImplForRefTemp
103            .anonymousLocalDataSourceImpl
104                .getCommunicationMessagesAnonymous()
105                    .then((value) => value.toList());
106
```

ถัดมาจะทำการเรียก getNotificationAndSortFromRealm ที่จะทำการ query realm จาก CommunicationMeassge และ CommunicationMessageMaster

```
100
101     List<CommunicationMessageAnonymous> allRefTempMessages =
102         await communicationMessageAnonymousRepositoryImplForRefTemp
103             .anonymousLocalDataSourceImpl
104                 .getCommunicationMessagesAnonymous()
105                     .then((value) => value.toList());
106
107     List<CommunicationMessage> sortData = await getNotiAndSortFromRealm();
108
109     for (int i = 0; i < sortData.length; i++) {
110         var noti = sortData[i];
111         if (noti.subject == null) {
112             continue;
113         }
114     }
```

### [getNotificationAndSortFromRealm](#)

การทำงานภายใน เริ่มจาก ทำการดึงค่า createdUserDatetime จาก getOldestCreateDateUser ที่อยู่ใน check login Service

เพื่อนำไป build query ส่าหรับดึงmessage ที่ถูกส่งมาใหม่กว่า วันที่ user เริ่มใช้งาน app  
ถัดมาจะทำการ ดึง default\_category\_push\_notification ใน constacnt เพื่อเลือกเฉพาะ category ที่กำหนดเท่านั้น หลังจากนั้นจะทำการประกอบ query ด้วยกัน เพื่อใช้ในการ query data ผ่าน LocalDataSourceImpl.queryCommunicationMessages(query) เพื่อดึง CommunicationMessages จาก realm และแบ่งลงให้กับราย [CommunicationMessage](#) list ธรรมด้า

```
821     Future<List<CommunicationMessage>> getNotiAndSortFromRealm() async {
822         String createdUserDatetime = DateFormat('yyyy-MM-dd@HH:mm:ss:ms', 'en')
823             .format(await getOldestCreateDateUser());
824         String categoryQuery =
825             " (${default_category_push_notification.join(",")}) ";
826         RealmResults<CommunicationMessage> resultGetAllNotifications =
827             await localDataSource.queryCommunicationMessages(
828                 "(sendTime >= $createdUserDatetime OR sendTime = null) AND (TORO_notificationCategory IN $categoryQuery OR TORO_notificationCategory = null)");
829
830         List<CommunicationMessage> sortData = resultGetAllNotifications.toList();
831     }
```

ถัดมาจะทำการ สร้าง tempNotiSortdata รอไว้ จากนั้นทำการวนลูป

ภายในลูปจะทำการ check ว่ามี noti ในลูปปัจจุบันซ้ำกับ noti ที่มีใน tempNotiSortdata หรือไม่

\*โดยกรณีที่ noti ซ้ำ จะทำการเช็คว่า **TORO\_notificationType** เป็น broadcast หรือไม่  
 เพราะกรณีที่เป็น broadcast จะหมายถึง noti ตัวนั้นเป็น refferent state ของการ อ่านหรือลบ จาก communicationMessageMaster อีกที ถัดมาเมื่อเช็คแล้วว่าเป็น type broadcast จะเช็คต่อว่า state เป็น cancelled หรือไม่ ถ้าใช่จะทำการอัปเดต state ของ noti ที่อยู่ tempNotiSortdata ตาม index ที่ข้อมูลซ้ำ และ ข้ามขั้นตอนอื่นๆ ไป

```

832     //merge duplicate noti
833     List<CommunicationMessage> tempNotiSortdata = [];
834
835     for (CommunicationMessage noti in sortData) {
836         //key to find this TORO_messageId
837         final toroId = noti.TORO_messageId;
838
839         //find index of has seen
840         int dupIndex = tempNotiSortdata
841             .indexWhere((element) => element.TORO_messageId == toroId);
842
843         //if found duplicate in temp
844         if (dupIndex != -1) {
845             if (noti.TORO_notificationType == "broadcast") [
846                 if (tempNotiSortdata[dupIndex].state != 'cancelled' &&
847                     noti.state == 'cancelled') {
848                         tempNotiSortdata[dupIndex].state = noti.state;
849                     }
850
851             continue;
852         }

```

ในกรณี TORO\_notificationType ไม่เป็น broadcast จะทำการแก้ค่า targetAsset จาก characteristic ใน noti ลุปปั๊บจุบัน และทำการเพิ่ม targetAsset เข้าไปใน targetAsset ของ tempNotiSortdata ตาม index ที่ข้อมูลซ้ำ โดยจะขันด้วย “|” ในแต่ละตัว ถ้ามาจะทำการเช็ค state ของตัว noti และ ถ้าหากค่าไม่เท่ากับ inprogress หรือยังไม่อ่าน จะทำการเขียนค่าตาม state ลงบน tempNotiSortdata ตาม index ที่ข้อมูลซ้ำ

```

853     //get newtarget
854     var newTargetAsset = noti.characteristic
855         .firstWhere((element) => element.name == 'targetAsset');
856
857     //find index of targetAsset in characteristic
858     int targetAssetIndex = tempNotiSortdata[dupIndex]
859         .characteristic
860             .indexWhere((element) => element.name == 'targetAsset');
861
862     //if characteristic not content targetAsset
863     if (targetAssetIndex == -1) {
864         //set targetAsset in temp = newTarget
865         tempNotiSortdata[dupIndex].characteristic[targetAssetIndex].value =
866             newTargetAsset.value;
867
868     //if characteristic content targetAsset
869     if (targetAssetIndex != -1) {
870         //combine targetAsset
871         var targetAA = tempNotiSortdata[dupIndex]
872             .characteristic[targetAssetIndex]
873             .value;
874
875         tempNotiSortdata[dupIndex].characteristic[targetAssetIndex] =
876             Characteristic(
877                 name: "targetAsset",
878                 value: "$targetAA ${newTargetAsset.value}");
879
880     if (noti.state != null && noti.state.toLowerCase() == "inprogress") {
881         tempNotiSortdata[dupIndex].state = noti.state;
882     }

```

\*และโดยกรณีที่ noti ไม่ซ้ำ จะทำการสร้าง object communicationMessage ตัวใหม่ขึ้นมาแล้ว add ลงใน tempNotiSortdata และเมื่อทำงานครบจำนวนทั้งหมดที่มีแล้ว จะทำการ clear ค่า sort data และแทนที่ด้วย tempNotiSortdata

```

882     } else {
883         CommunicationMessage uniqueNoti = CommunicationMessage(
884             noti.id,
885             noti.uuid,
886             noti.ts,
887             content: noti.content,
888             description: noti.description,
889             logFlag: noti.logFlag,
890             messageType: noti.messageType,
891             priority: noti.priority,
892             scheduledSendTime: noti.scheduledSendTime,
893             sendTime: noti.sendTime,
894             sendTimeComplete: noti.sendTimeComplete,
895             subject: noti.subject,
896             state: noti.state,
897             validFor: noti.validFor,
898             sender: noti.sender,
899             TORO_notificationType: noti.TORO_notificationType,
900             TORO_notificationCategory: noti.TORO_notificationCategory,
901             TORO_messageId: noti.TORO_messageId,
902             isTemp: noti.isTemp,
903             localizedString: noti.localizedString,
904             attachment: noti.attachment,
905             characteristic: noti.characteristic,
906             receiver: noti.receiver,
907         );
908
909         tempNotiSortdata.add(uniqueNoti);
910     }
911
912     sortData.clear();
913     sortData.addAll(tempNotiSortdata);
914
915

```

ถ้ามีการทำ query ตัว CommunicationMessageMaster ด้วย query string ที่เหมือนกับตอนที่ query ตัว CommunicationMessage หลังจากนั้นจะทำการเช็ค result ว่ามี noti หรือไม่ถ้ามีจะทำการลูปตาม result ที่ได้รับมาโดยภายใน loop จะทำการสร้าง object communicationMessage ด้วย property ของ CommunicationMessageMaster

```

916     RealmResults<CommunicationMessageMaster>
917     resultGetAllBroadCastNotifications =
918     await localDataSource.queryCommunicationMessagesMaster(
919         "sendTime >= screatedUserDateTime AND (T00_notificationCategory IN $categoryQuery OR T00_notificationCategory = null)");
920
921     if (resultGetAllBroadCastNotifications.isNotEmpty) {
922         var broadcasts = resultGetAllBroadCastNotifications.toList();
923         for (CommunicationMessageMaster notiOnMaster = CommunicationMessage(
924             broadcastNoti.id,
925             broadcastNoti.uuid,
926             broadcastNoti.ts,
927             content: broadcastNoti.content,
928             description: broadcastNoti.description,
929             logLog: broadcastNoti.logLog,
930             messageType: broadcastNoti.messageType,
931             priority: broadcastNoti.priority,
932             scheduledSendTime: broadcastNoti.scheduledSendTime,
933             sendTime: broadcastNoti.sendTime,
934             sendTimeComplete: broadcastNoti.sendTimeComplete,
935             subject: broadcastNoti.subject,
936             tryTimes: broadcastNoti.tryTimes,
937             state: broadcastNoti.state,
938             validFor: broadcastNoti.validFor,
939             sender: broadcastNoti.sender,
940             T00_notificationType: broadcastNoti.T00_notificationType,
941             T00_notificationCategory: broadcastNoti.T00_notificationCategory,
942             T00_messageId: broadcastNoti.T00_messageId,
943             isTemp: broadcastNoti.isTemp,
944             localizedString: broadcastNoti.localizedString,
945             attachment: broadcastNoti.attachment,
946             characteristic: broadcastNoti.characteristic,
947             receiver: broadcastNoti.receiver,
948         );
949     );
950

```

หลังจากนั้นจะทำการเช็คว่า CommunicationMessageMaster ในลูปปัจจุบันมีที่ noti ในตัว sortdata ที่เป็น CommunicationMessage หรือไม่ โดยถ้ามี จะเช็คต่อว่ามี state เท่ากับ cancelled หรือไม่ถ้าใช่จะทำการลบ ข้อมูลชุดนั้นออกจาก sortdata แต่ถ้าไม่เท่ากับ cancelled จะทำการปรับข้อมูลบางส่วนแล้วนำไปเขียนทับบน sortdata

กรณีที่ไม่พบว่ามีตัว noti อยู่ใน sortData อยู่ก่อน จะทำการเพิ่ม noti นั้นเข้าไป

เมื่อวนจนครบแล้วจะทำการคัด noti ใน sortData ที่มี state เป็น cancelled และ ไม่มี content จากนั้นทำการจัดเรียงข้อมูล ตามวันที่ส่งล่าสุดไปเก่าสุดโดยถ้าไม่มีวันส่งจะใช้ค่า ts แทน

```

951     int index = sortData.indexWhere((element) =>
952         element.T00_messageId == broadcastNoti.T00_messageId);
953
954     if (index != -1) {
955         if (sortData[index].state != "cancelled") {
956             notiOnMaster.id = sortData[index].id;
957             notiOnMaster.ts = sortData[index].ts;
958             notiOnMaster.state = sortData[index].state;
959             sortData[index] = notiOnMaster;
960         } else {
961             sortData.removeAt(index);
962         }
963     } else {
964         sortData.add(notiOnMaster);
965     }
966 }
967
968 sortData.removeWhere((element) =>
969     element.state == "cancelled" ||
970     (element.T00_notificationType == "broadcast" &&
971      element.content == null));
972
973 sortData.sort((a, b) {
974     if (a.sendTime != null && b.sendTime != null) {
975         return b.sendTime!.compareTo(a.sendTime!);
976     } else {
977         return b.ts.timestamp.compareTo(a.ts.timestamp);
978     }
979 });
980
981 }
982

```

หลังจากที่ได้ sortData ก็ลับมาแล้วจะทำการวนลูป ข้อมูลโดยในลูปอันดับแรกจะทำการเช็ค noti ตัวนั้นๆ ก่อนว่ามี subject ที่เป็นหัวข้อของ noti หรือไม่ถ้าไม่มีจะทำการข้ามไป

ถ้ามาจะทำการเช็คว่ามีการ filterAsset หรือไม่ ถ้ามีจะทำการค้นหาว่า noti นั้นๆ ว่ามี assett ที่ตรงกับ filter หรือไม่ ถ้าไม่มีจะทำการข้าม

```
107     List<CommunicationMessage> sortData = await getNotiAndSortFromRealm();
108
109     for (int i = 0; i < sortData.length; i++) {
110         var noti = sortData[i];
111         if (noti.subject == null) {
112             continue;
113         }
114
115         //filterAsset
116         if (filterAsset != null && noti.characteristic.isNotEmpty) {
117             //get asset from noti on characteristic
118             var targetAssetOnNoti = noti.characteristic.firstWhere(
119                 (element) => element.name == 'targetAsset',
120                 orElse: () =>
121                     return Characteristic(value: ""),
122             );
123             if (!targetAssetOnNoti.value
124                 .toString()
125                 .contains(filterAsset.assetValue)) {
126                 continue;
127             }
128         }
129     }
130 }
```

ถ้ามาจะทำการเช็ค ข้อมูลใน allRefTempMessages ว่ามี id ที่ตรงกันอยู่หรือไม่  
ถ้ามี จะทำการเปลี่ยน state ว่าถ้าเป็น cancelled จะทำการข้าม noti ตัวนั้นไป  
หรือ ถ้าเป็น read จะทำการเขียนค่า read ตาม refferent จากนั้นทำการ buildNotificationContent  
ให้กลายเป็น entity model

จากนั้นทำการ map push category ไปเป็น logical category ผ่าน constanct map  
จากนั้นทำการเช็ค ว่าได้ logical category กับมาหรือไม่ ถ้ามีทำการเช็ค property isRead  
เพิ่มนำไป set ค่า badge หลังจากนั้นทำการเพิ่มลงไปใน map object ที่เตรียมไว้

```
146     //check category
147     String? logicalCategory = default_category_push_notification_mapping[
148         notificationContentBuilded.category];
149     if (logicalCategory != null) {
150         //add badge
151         if (!notificationContentBuilded.isRead) {
152             baged["all"]! = baged["all"]! + 1;
153             baged[logicalCategory] = baged[logicalCategory]! + 1;
154         }
155
156         //notification
157         notificationList["all"]!.add(
158             BehaviorSubject<NotificationModel>.seeded(
159                 notificationContentBuilded));
160         notificationList[logicalCategory]!.add(
161             BehaviorSubject<NotificationModel>.seeded(
162                 notificationContentBuilded));
163     }
164
165
166     //build last result
167     //add all first
168     Map<String, NotificationEntity> resultAllNotification = {
169         "all": NotificationEntity(
170             category: "all",
171             badge: BehaviorSubject<int>.seeded(baged["all"]!),
172             notifications:
173                 BehaviorSubject<List<BehaviorSubject<NotificationModel>>>.seeded(
174                     notificationList["all"]!, // BehaviorSubject.seeded
175                 ) // NotificationEntity
176     };
177 }
```

สุดท้ายนำ map object ของ result ทั้งหมดประกอบกันเป็น NotificationEntity และวนนำไปใส่ใน map NotificationEntity อีกครั้งและคืน result กลับไป

```
166     //build last result
167     //add all first
168     Map<String, NotificationEntity> resultAllNotification = {
169         "all": NotificationEntity(
170             category: "all",
171             badge: BehaviorSubject<int>.seeded(baged["all"]!),
172             notifications:
173                 BehaviorSubject<List<BehaviorSubject<NotificationModel>>>.seeded(
174                     notificationList["all"]!), // BehaviorSubject.seeded
175                 ) // NotificationEntity
176         };
177
178     //add following logicalCategory
179     for (String logicalCategory in default_category_notifiaction) {
180         resultAllNotification.addAll({
181             logicalCategory: NotificationEntity(
182                 category: logicalCategory,
183                 badge: BehaviorSubject<int>.seeded(baged[logicalCategory]!),
184                 notifications: BehaviorSubject<
185                     List<BehaviorSubject<NotificationModel>>>.seeded(
186                         notificationList[logicalCategory]!), // BehaviorSubject.seeded
187                     ) // NotificationEntity
188                 });
189     }
190     return Right(resultAllNotification);
```

กรณีที่ user ยังไม่ได้ login

จะใช้งาน anonymousRepositoryImpl.getAllNotification() ซึ่งมี process คล้ายกับ ตอน login  
แต่จะทำการไม่ทำการ query CommunicationMessage ขึ้นมา

```
51     Map<String, int> baged = {"all": 0};
52     Map<String, List<BehaviorSubject<NotificationModel>>> notificationList = {
53         "all": []
54     };
55
56     for (String logicalCategory in default_category_notifiaction) {
57         //baged
58         baged.addAll({logicalCategory: 0});
59         //notification
60         notificationList.addAll({logicalCategory: []});
61     }
62
63     List<CommunicationMessage> sortData = [];
64     List<CommunicationMessageAnonymous> anonymousNoti =
65         (await anonymousLocalDataSourceImpl
66             .getCommunicationMessagesAnonymous())
67             .toList();
68
69     final prefs = await SharedPreferences.getInstance();
70     String? userInstallAppDatetime =
71         prefs.getString(prefs_key_installAppDatetime);
72
73     String categoryQuery =
74         " ${default_category_push_notifiaction.join(',')}"';
75
76     RealmResults<CommunicationMessageMaster>
77         result GetAllBroadCastNotifications =
78         await localDataSource.queryCommunicationMessagesMaster(
79             "sendTime >= $userInstallAppDatetime AND (T00_notificationCategory IN $categoryQuery)");
```

## การดึงค่า badge ข้อความที่ยังไม่อ่าน

การดึง badge ข้อความที่ไม่ได้อ่าน จะทำการ เรียกผ่าน usecase getBadgesUnReadMessages โดย process จะเริ่มจากการ เช็ค login ก่อน เพื่อเลือก repository ตาม role login

```
34 Future<Either<Failure, int>> getBadgesUnReadMessages() async {  
35     var islogin = await isLogin();  
36     Either<Failure, int> badgeResult;  
37     if (islogin) {  
38         badgeResult = await repository.getBadgesUnReadMessages();  
39     } else {  
40         badgeResult = await anonymousRepositoryImpl.getBadgesUnReadMessages();  
41     }  
42 }
```

### กรณี login

จะใช้งาน CommunicationRepositoryImpl getBadgesUnReadMessages โดยจะทำ query allRefTempMessages จาก CommunicationMessageAnonymous ไว้สำหรับเที่ยบ action ล่าสุด ถัดมาจะเรียกใช้งาน [getNotificationAndSortFromRealm](#) เมื่อได้ sortData กลับมาจะทำการวนลูป เพื่อเทียบ action ล่าสุด และทำการนับ state ของ noti ที่ยังไม่อ่าน เมื่อทำจนครบจะทำการลับไป

```
196     @override  
197     Future<Either<Failure, int>> getBadgesUnReadMessages() async {  
198         try {  
199             int badges = 0;  
200             List<CommunicationMessageAnonymous> allRefTempMessages =  
201                 await communicationMessageAnonymousRepositoryImplForRefTemp  
202                     .anonymousLocalDataSourceImpl  
203                     .getCommunicationMessagesAnonymous()  
204                     .then((value) => value.toList());  
205  
206             List<CommunicationMessage> sortData = await getNotiAndSortFromRealm();  
207  
208             for (int i = 0; i < sortData.length; i++) {  
209                 var noti = sortData[i];  
210                 try {  
211                     CommunicationMessageAnonymous refTempMessage = allRefTempMessages  
212                         .firstWhere(element => element.id == noti.TORO_messageId!);  
213                     if (refTempMessage.state == "cancelled") {  
214                         continue;  
215                     }  
216                     if (refTempMessage.state == "read") {  
217                         noti.state = "read";  
218                     }  
219                     } catch (e) {  
220                         //  
221                     }  
222  
223                     if (noti.state != "read") {  
224                         badges += 1;  
225                     }  
226                 }  
227             }  
228         }  
229     }
```

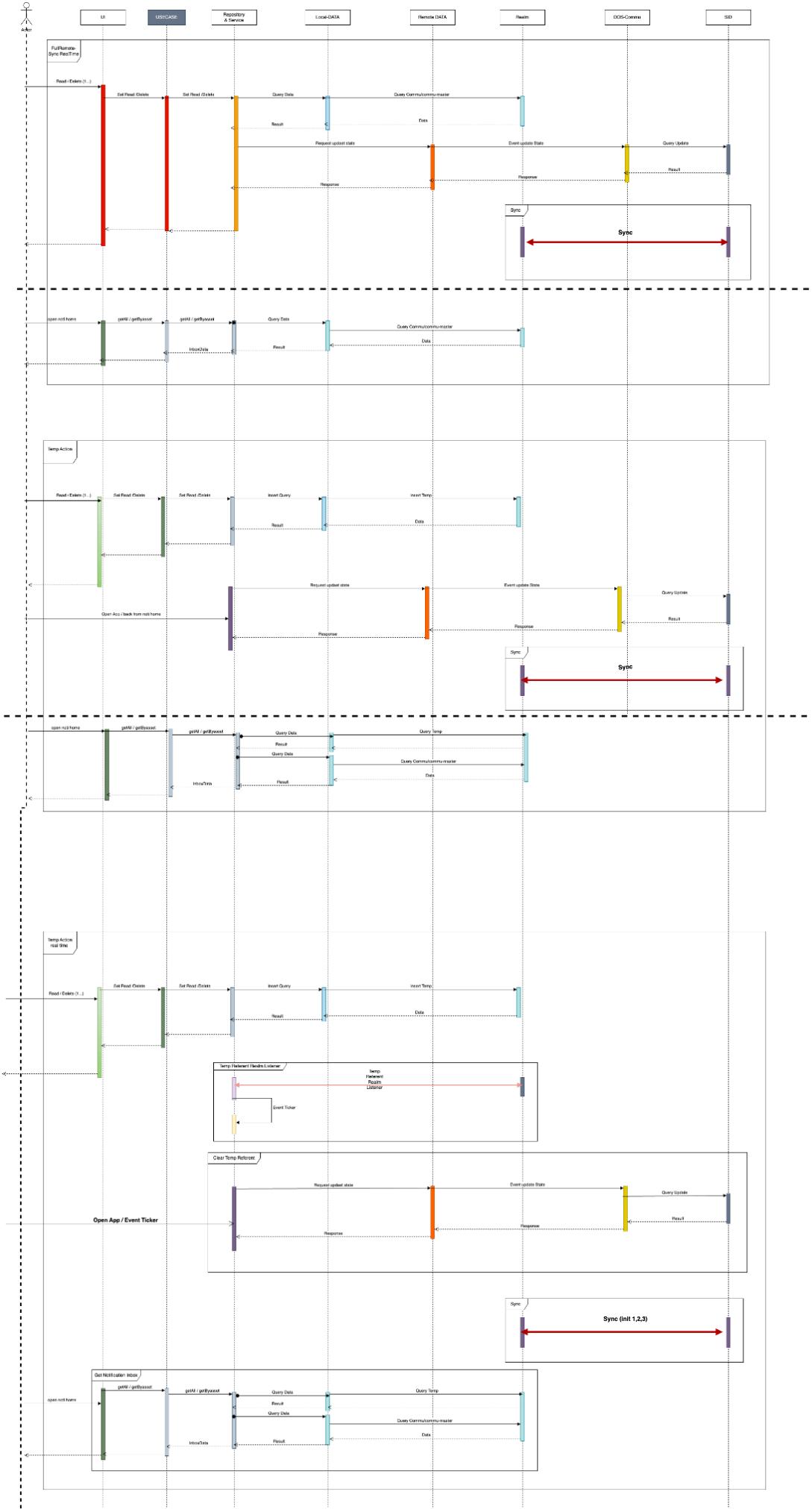
### กรณี non login

จะใช้งาน CommunicationMessageAnonymousRepositoryImpl getBadgesUnReadMessages โดยจะทำ query CommunicationMessagesMaster และ CommunicationMessageAnonymous ที่เป็น message broadcast และ last action user และทำการหักลบกัน จะได้ข้อความที่ยังไม่ถูกอ่าน และคืนค่ากลับไป

```
288     @override  
289     Future<Either<Failure, int>> getBadgesUnReadMessages() async {  
290         try {  
291             final prefs = await SharedPreferences.getInstance();  
292             String? userInstallAppDatetime =  
293                 prefs.getString(prefs_key_installAppDatetime);  
294  
295             String categoryQuery =  
296                 " ${defult_category_push_notifiaction.join(',')}" ;  
297  
298             var MasterMsg = await localDataSource.queryCommunicationMessagesMaster(  
299                 "sendTime >= $userInstallAppDatetime AND (TORO_notificationCategory IN $categoryQuery)");  
300  
301             var AnonymousMsg = await anonymousLocalDataSourceImpl  
302                 .getCommunicationMessagesAnonymous();  
303             return Right(MasterMsg.length - AnonymousMsg.length);  
304         } catch (e) {  
305             return Left(ServerFailure(e.toString()));  
306         }  
307     }  
308 }
```

หลังจากได้ค่า badge กลับมา จะทำการเช็ค permision ของ device เปิดการแจ้งเตือนหรือไม่ ถ้าเปิดจะทำการเช็คว่า device รองรับการอัปเดตค่าข้อความที่โอลดอนแอพหรือไม่ ถ้ารองรับจะทำการ set ค่าลงไป และนำ ค่า badge ไปฝากที่ SharedPreferences เพื่อใช้งานอีกครั้ง

```
43     badgeResult.fold((l) => null, (r) async {
44       //try to set app icon badge
45       try {
46         String notiPremision =
47           await PushNotificationService().getNotiPermission();
48         if (notiPremision == "enable") {
49           CoreLog().info(
50             "Try to set app icon badge to $r",
51           );
52           var spb = await FlutterAppBadger.isAppBadgeSupported();
53           CoreLog().info(
54             "Try to set app icon badge isAppBadgeSupported: $spb",
55           );
56           FlutterAppBadger.updateBadgeCount(r);
57         }
58         final prefs = await SharedPreferences.getInstance();
59         prefs.setInt(prefs_key_AppBadge, r);
60       } catch (e, strak) {
61         CoreLog().error(
62           "Try to set app icon badge failed",
63           error: e,
64           stackTrace: strak,
65         );
66       }
67     });
68   }
69   return badgeResult;
```



## การอ่านข้อความใน inbox

การอ่านข้อความใน inbox จะทำการเรียกใช้งาน usecase setReadedMessagesByID ที่จะรับ parameter ที่จำเป็นคือ id ของ message และ myId ที่ต้องการลบ โดย process จะเริ่มจากการ เช็ค login ก่อน เพื่อเลือก repository ตาม role login

```
72 Future<Either<Failure, void>> setReadedMessagesByID(
73     @required String msg_id,
74     @required String myid,
75     String? usecaseName,
76     String? usecaseStep)) async [
77     var isLogin = await isLogin();
78     if (isLogin) {
79         CoreLog().info("setReadedMessagesByID user");
80         return repository.setReadedMessagesByID(
81             msg_id: msg_id,
82             myid: myid,
83             usecaseName: usecaseName,
84             usecaseStep: usecaseStep,
85         );
86     } else {
87         CoreLog().info("setReadedMessagesByID anonymous");
88         return anonymousRepositoryImpl.setReadedMessagesByID(msg_id: msg_id);
89     }
90 ]
```

### กรณี login

จะใช้งาน CommunicationRepositoryImpl setReadedMessagesByID โดยจะทำ query getCommunicationMessagesByID จาก CommunicationMessage ผ่าน LocalDataSourceImpl เมื่อได้ result และทำการเช็คว่าอ่านไปหรือไม่และเป็น type ของ individual หรือไม่  
**แต่ถ้าไม่ได้ result** จะ query getCommunicationMessagesMasterByID แทน  
ถ้าผ่านเงื่อนไขจากข้อแรก หรือมี result ในข้อสอง จะทำการ เรียกใช้ communicationMessageAnonymousRepositoryImplForRefTemp.setReadedMessagesByID โดยส่ง id ที่ใช้ค่าจาก TORO\_messageID แบบไป

```
76 Future<Either<Failure, void>> setReadedMessagesByID(
77     @required String msg_id,
78     @required String myid,
79     String? usecaseName,
80     String? usecaseStep)) async {
81     try {
82         CoreLog().info("setReadedMessagesByID ID: $msg_id");
83         var noti = null;
84         try {
85             noti = await localDataSource.getCommunicationMessagesByID(msg_id);
86             if (noti != null && noti.notificationType.toLowerCase() == "read" &&
87                 noti.TORO_notificationType.toLowerCase() == "individual") {
88                 CoreLog().info(
89                     "Noti DATA notatory: correlationId: ${noti.correlationId}, TORO_messageId: ${noti.TORO_messageId}, TORO_NotificationType: ${noti.notificationType}, communicationMessageAnonymousRepositoryImplForRefTemp
90                     .setReadedMessagesByID(msg_id: noti.TORO_messageId);");
91             }
92         } catch (e) {
93             noti = null;
94         }
95     }
96     try {
97         noti = await localDataSource.getCommunicationMessagesMasterByID(msg_id);
98         if (noti != null) {
99             communicationMessageAnonymousRepositoryImplForRefTemp
100                 .setReadedMessagesByID(msg_id: noti.TORO_messageId);
101         }
102     } catch (e) {
103     }
104 }
```

### กรณี non-login

จะใช้งาน communicationMessageAnonymousRepositoryImplForRefTemp.setReadedMessagesByID โดยส่ง id ที่ใช้ค่าจาก TORO\_messageID แบบไป

### *communicationMessageAnonymousRepositoryImplForRefTemp.setReadedMessagesByID*

ท่านน้าที่ในการ insert read action ที่ใช้เป็น Reffernt Message โดยการทำงานจะเริ่มจากการเช็คว่ามี insert ไปแล้วหรือยัง โดยถ้ายังไม่มี จะทำการ เพิ่มข้อมูลตัวใหม่ลงไปโดยใช้ message id และ state เป็น read และใช้ CommunicationMessageAnonymousLocalDataSourceImpl insertUpdateWithModelCommunicationMessageAnonymous เพื่อเพิ่มลงใน realm

```

309     @override
310     Future<Either<Failure, void>> setReadedMessagesByID(
311         required String msg_id) async {
312     try {
313         CoreLog().info("setReadedMessagesByID ID: $msg_id");
314         var noti = null;
315         try {
316             noti = await anonymousLocalDataSourceImpl
317                 .getCommunicationMessagesAnonymousById(msg_id);
318         } catch (e) {
319             CoreLog().error(
320                 "setReadedMessagesByID exception",
321                 error: e,
322             );
323             noti = null;
324         }
325
326         if (noti == null) {
327             CommunicationMessageAnonymous newRefNotiAnonymous =
328                 CommunicationMessageAnonymous(msg_id, msg_id, "read");
329
330             await anonymousLocalDataSourceImpl
331                 .insertUpdateWithModelCommunicationMessageAnonymous(
332                     newRefNotiAnonymous);
333         }
334
335         return const Right(null);
336     } catch (e) {
337         //throw UnimplementedError();
    }

```

### การอ่านข้อความจากการกด notification

การอ่านข้อความจากการกด notification จะใช้ usecase setReadedMessagesByToroMessageID  
เนื่อง payload ของ notificatyon จะไม่ได้ส่งตัว id ที่เป็น key ใน sid และ realmมาให้ จึงใช้ ToroMessageID จาก payload ของ notificationแทน โดยตัว usecase จะใช้งาน setReadedMessagesByToroMessageID และ ใช้งาน communicationMessageAnonymousRepositoryImplForRefTemp.setReadedMessagesByID โดยส่ง id ที่ใช้ค่าจาก TORO\_messageid แบบไป

```

92     Future<Either<Failure, void>> setReadedMessagesByToroMessageID(
93         required String toroMessageId,
94         required String toroNotificationType,
95         required String myid,
96         String? usecaseName,
97         String? usecaseStep) async {
98         //var islogin = await islogin();
99         //CoreLog().info("setReadedMessagesByToroMessageID ");
100        return repository.setReadedMessagesByToroMessageID[
101            toroMessageId: toroMessageId,
102            toroNotificationType: toroNotificationType,
103            myid: myid,
104            usecaseName: usecaseName,
105            usecaseStep: usecaseStep,
106        ];
107    }
    @override
18     Future<Either<Failure, void>> setReadedMessagesByToroMessageID(
19         required String toroMessageId,
20         required String toroNotificationType,
21         required String myid,
22         String? usecaseName,
23         String? usecaseStep) async {
24     try {
25         CoreLog().info(
26             "setReadedMessagesByToroMessageID deatil: TORO_messageId:$toroMessageId, TORO_notificationType:$toroNotificationType");
27         communicationMessageAnonymousRepositoryImplForRefTemp
28             .setReadedMessagesByID(msg_id: toroMessageId);
    }

```

## การลบข้อความใน INBOX

### การดึงค่าการตั้งค่าของ Notification

การดึง setting ของ Notificaiton จะทำผ่าน usecase getAllowedNotificationCategory และเรียกใช้งาน CommunicationRepositoryImpl getAllowedNotificationCategory

```
130     Future<Either<Failure, dynamic>> getAllowedNotificationCategory(String myid,
131         {String? usecaseName, String? usecaseStep}) {
132         return repository.getAllowedNotificationCategory(
133             myid,
134             usecaseName: usecaseName,
135             usecaseStep: usecaseStep,
136         );
137     }
```

การทำงานของ getAllowedNotificationCategory จะทำการสร้าง map result ที่ใส่ค่า category, serivecInvenId ว่างไว้ก่อน และทำการดึงค่า setting จาก SharedPreferences ด้วย key

prefs\_key\_allowedNotificationCategory โดยถ้าไม่มี result จะกำหนดค่าเป็น [“-”]

หลังจากนั้นจะทำการ update map result

```
315     @Override
316     Future<Either<Failure, dynamic>> getAllowedNotificationCategory(String myid,
317         {String? usecaseName, String? usecaseStep}) async {
318         List<String> cat = [];
319         String serviceInvenID = "";
320         var result = {
321             "category": cat,
322             "serviceInvenID": serviceInvenID,
323         };
324
325         //do local
326         try {
327             CoreLog().info(
328                 " on try get SharedPreferences prefs_key_allowedNotificationCategory");
329             final prefs = await SharedPreferences.getInstance();
330             var on_prefs = prefs.getStringList(prefs_key_allowedNotificationCategory);
331             if (on_prefs != null) {
332                 cat = on_prefs;
333                 CoreLog().info("on try get SharedPreferences result : $cat");
334             } else {
335                 cat = ["-"];
336             }
337
338             result = {
339                 "category": cat,
340                 "serviceInvenID": serviceInvenID,
341             };
342         } catch (e) {
343             return Left(ServerFailure(e.toString()));
344         }
345     }
```

ถัดมาจะทำการ query ProductInventory ผ่าน RemoteDataSource queryProductInventory โดยจะส่ง payload = { "relatedParty": [myid] } ไปเพื่อใช้ในการขอ request

```
//try remote
try {
    var productInven = {
        "relatedParty": [myid]
    };
    var productInven_result = await remoteDataSource.queryProductInventory(
        productInven,
        usecaseName: usecaseName,
        usecaseStep: usecaseStep,
    );
}
```

## RemoteDataSource queryProductInventory

เมื่อรับ parameter มาแล้วจะทำการบันน์ base haeder ,coreBaseOption สำหรับ queryProductInventory และทำการ requestNetwork ผ่าน socketHepler และส่ง response ศีนกลับไป

```
160    @Override
161    Future<Map<String, dynamic>> queryProductInventory(Map<String, dynamic> productInventory,
162        String? usecaseName, String? usecaseStep) async {
163        Map<String, String> baseHeaders = {
164            "Accept": "application/json",
165            "Content-type": "application/json; charset=utf-8",
166        };
167        baseHeaders.addAll({
168            HeaderKey.topic.name: "mfaf.queryProductInventory",
169            HeaderKey.orgService.name: "common.communication",
170            HeaderKey.tmfSpec.name: "TMF637",
171            HeaderKey.baseApiVersion.name: '4.0.0',
172            HeaderKey.eventResponse.name:
173                "mfaf.productInventoryQueried,mfaf.queryProductInventoryFailed",
174        });
175
176        try {
177            CoreBaseOption coreBaseOption = CoreBaseOption(
178                useCase: usecaseName,
179                useCaseStep: usecaseStep,
180                headers: baseHeaders,
181                body: productInventory,
182                timeout: Duration(seconds: 10),
183            ); // CoreBaseOption
184            CoreLog().info(
185                '+++++queryProductInventory body request+++++++$({productInventory})');
186            var response = await socketHelper.requestNetwork(coreBaseOption);
187
188            CoreLog().info(
189                '=====queryProductInventory response datasource===== ${response}');
190            return response;
191        } catch (e) {
192            rethrow;
193        }
194    }
195 }
```

จากนั้นจะใช้ body ของ response และทำการแกะ productInventory จาก body อีกที่ และนำมารวนลูป เพื่อรวม realizingService เป็นก้อนเดียว หลังจากนั้นจะทำการวนลูปจากก้อน realizingService โดยจะวนลูปแบบกลับหลังเพื่อหา realizingService ที่มี name เป็น ของ Notification ถ้าเจอก็จะทำการเก็บค่า id ซึ่งเป็นค่า serviceInvenID และทำการ อัปเดตค่า map result

```
357    var productInven_body_result = productInven_result["body"];
358
359    var realizingService = [];
360
361    for (var productInventory
362        in productInven_body_result["productInventory"]) {
363        // if (productInventory["realizingService"]){}
364        var rzs = productInventory["realizingService"];
365        realizingService = realizingService + rzs;
366    }
367
368    CoreLog().info("realizingService : $realizingService");
369    int i = realizingService.length - 1;
370
371    for (i; i >= 0; i--) {
372        if (realizingService[i]["name"] == "Notification") {
373            serviceInvenID = realizingService[i]["id"];
374            break;
375        }
376    }
377
378    CoreLog().info(
379        "productInven_result body:" + productInven_body_result.toString());
380
381    CoreLog().info("serviceInven ID : $serviceInvenID");
382
383    result = [
384        {"category": cat,
385         "serviceInvenID": serviceInvenID},
386    ];
387
388    //return Right(result);
389    } catch (e) {
390        CoreLog().error(
391            "on get remote allowed category",
392            error: e,
393        );
394    }
395 }
```

จากนั้นจะทำการ map push category ไปเป็น logical category ผ่าน mapping constant และทำการคืนค่า

```
395 // use local category , use service id
396 List<String> tempCat = [];
397 for (String pushCat in cat) {
398     String? logicCat =
399         defult_category_push_notifiaction_mapping[pushCat.toString()];
400     if (logicCat != null && tempCat.contains(logicCat) == false) {
401         tempCat.add(logicCat);
402     }
403 }
404
405 cat = tempCat;
406 result = {
407     "category": cat,
408     "serviceInvenID": serviceInvenID,
409 };
410 return Right(result);
411 }
```

## การแก้ไขการตั้งค่าของ Notification

การแก้ไขการตั้งค่า notification จะทำผ่าน usecase setAllowedNotificationCategory ซึ่งจะรับ parameter list category และ serviceInventID ที่ได้จาก usecase getAvailableNotificationCategory

โดยจะเรียก CommunicationRepositoryImpl setAllowedNotificationCategory เพื่อทำงานต่อ

```
139 Future<Either<Failure, void>> setAllowedNotificationCategory(
140     {required List<String> category,
141      required String serviceInvenID,
142      String? usecaseName,
143      String? usecaseStep}) {
144     return repository.setAllowedNotificationCategory(
145         category: category,
146         serviceInvenID: serviceInvenID,
147         usecaseName: usecaseName,
148         usecaseStep: usecaseStep,
149     );
150 }
```

ใน CommunicationRepositoryImpl setAllowedNotificationCategory จะเริ่มต้นทำงานจากการแปลง logical category to push category และทำการเรียก updateCategoryOnDevice

```
444 @override
445 Future<Either<Failure, void>> setAllowedNotificationCategory(
446     {required List<String> category,
447      required String serviceInvenID,
448      String? usecaseName,
449      String? usecaseStep}) async {
450     try {
451         category = mapLogicalCategoryToPushCategory(category).toSet().toList();
452         await updateCategoryOnDevice(category);
453         // var serviceInvenID = "9999";
454         var serviceInven = {"id": serviceInvenID};
455         var serviceInven_result = await remoteDataSource.queryServiceInventory(
456             serviceInven,
457             usecaseName: usecaseName,
458             usecaseStep: usecaseStep,
459         );
460 }
```

## mapLogicalCategoryToPushCategory

จะทำหน้าที่เป็นตัวแปลง จาก logical category ที่เป็น category สำหรับแสดงบนหน้าแอป ไปเป็น push category ที่เป็น category ที่ใช้ในการส่ง notification โดยจะทำการเทียบผ่าน constant ใน util

```
9  List<String> mapLogicalCategoryToPushCategory(List<String> category) {
10    List<String> result = [];
11    for (int i = 0; i < category.length; i++) {
12      if (category[i].toString() == "-") {
13        // if "-" use that (is disable)
14        result.add(category[i].toString());
15      } else {
16        // else, category maybe logical category, try to map to push category and use that
17        List<String>? temp;
18        temp = default_category_notification_mapping[(category[i].toString())];
19        if (temp != null) {
20          result.addAll(temp);
21        }
22
23        // else {
24        //   if (default_category_push_notification
25        //       .contains(category[i].toString())) {
26        //     // if category is one of push category use that
27        //     result.add(category[i].toString());
28        //   }
29        // }
30      }
31    }
32    return result;
33  }
34 }
```

## updateCategoryOnDevice

จะทำการอัปเดตการตั้งค่าลงบน SharedPreferences ด้วย key prefs\_key\_allowedNotificationCategory จากนั้น จะทำการวนลูปเช็คตาม default\_category\_push\_notification เพื่อที่จะทำการ เรียก PushNotificationService subscribeToTopicFCM , unsubscribeFromTopicFCM โดยถ้า category มีการส่งมา จะ sub ถ้าไม่ เจอก็จะทำการ unsub หลังจากนั้นจะทำการเช็คภาษา ณ ปัจจุบัน เพื่อทำการ subscribeToTopicFCM ตัวภาษาเพิ่มเข้าไป

```
179 Future<void> updateCategoryOnDevice(List<String> category) async {
180   //defaultCategory.contains(element)
181   final prefs = await SharedPreferences.getInstance();
182   //String fcmToken = PushNotificationService.FCM_TOKEN;
183   try {
184     await prefs.setStringList(prefs_key_allowedNotificationCategory, category);
185     for (int i = 0; i < default_category_push_notification.length; i++) {
186       if (category.contains(default_category_push_notification[i])) {
187         PushNotificationService()
188           .subscribeToTopicFCM(default_category_push_notification[i]);
189       } else {
190         PushNotificationService()
191           .unsubscribeFromTopicFCM(default_category_push_notification[i]);
192       }
193     }
194
195     await prefs.setString(
196       prefs_key_notification_lang, CurrentLang().currentLanguage);
197     // subscribe language ToTopicFCM
198     for (int i = 0; i < CurrentLang().localeList.length; i++) {
199       if (CurrentLang().currentLanguage == CurrentLang().localeList[i]) {
200         PushNotificationService()
201           .subscribeToTopicFCM(CurrentLang().currentLanguage);
202       } else {
203         PushNotificationService()
204           .unsubscribeFromTopicFCM(CurrentLang().localeList[i]);
205       }
206     }
207   } catch (e, stack) {
208     CoreLog().error(
209       "Exception on updateCategoryOnDevice",
210       error: e,
211       stackTrace: stack,
212     );
213   }
}
```

ถัดมาจะทำการ query ServiceInventory ผ่าน RemoteDataSource quiryServiceInventory โดยจะส่ง payload ={"id": serviceInventID}; ไปเพื่อใช้ในการขอ request  
 หลังจากนั้น จะนำเอา feature ที่อยู่ใน body ของ response ออกรมา พักไว้ที่ serviceFeature  
 ถัดมาจะทำการวนลูป serviceFeature เพื่อค้นหา name == notificationCategory  
 เมื่อเจอแล้วจะทำการวนลูปซ้ำ ในตัว เพื่อหา allowedNotificationCategory จาก name ของ featureCharacteristic จากนั้นทำการ update ค่า value ของ featureCharacteristic ของ serviceFeature ด้วย category ที่ส่งมา + ภาษาปัจจุบัน

```

454     var serviceInven = {"id": serviceInventID};
455     var serviceInven_result = await remoteDataSource.quiryServiceInventory(
456         serviceInven,
457         usecaseName: usecaseName,
458         usecaseStep: usecaseStep,
459     );
460
461     List<dynamic> serviceFeature = serviceInven_result["body"]["feature"];
462     CoreLog().info("original serviceInver : ${serviceInven_result["body"]}");
463     int i = 0;
464     for (i; i < serviceFeature.length; i++) {
465         CoreLog().info("ON serviceFeature :$i");
466         if (serviceFeature[i]["name"] == "notificationCategory") {
467             CoreLog().info("ON serviceFeature found notificationCategory @ $i");
468             List<dynamic> featureCharacteristic =
469                 | serviceFeature[i]["featureCharacteristic"];
470
471             int j = 0;
472             for (j; j < featureCharacteristic.length; j++) {
473                 CoreLog().info("ON serviceFeature featureCharacteristic: $j");
474                 if (featureCharacteristic[j]["name"] ==
475                     "allowedNotificationCategory") {
476                     CoreLog().info(
477                         | "ON serviceFeature featureCharacteristic found allowedNotificationCategory @ $j");
478                     //List<String>.from(featureCharacteristic[j]["value"] as List);
479                     serviceInven_result["body"]["feature"][i]["featureCharacteristic"]
480                         [j]["value"] = [CurrentLang().currentLanguage] + category;
481
482                     serviceInven_result["body"]["feature"][i]["featureCharacteristic"]
483                         [j]["lastModified"] =
484                             DateTime.now().toUtc().toIso8601String();
485                     CoreLog().info("featureCharacteristic break!!!!");
486                     break;
487                 }
488             }
489         }
490     }
491     CoreLog().info("serviceFeature break!!!!");
492     break;
493 }
494 }
495

```

ถัดมาทำการ อัปเดต startDate, endDate และทำการส่งกับไป update ผ่าน RemoteDataSource queryProductInventory เป็นอันจบ

```

495
496     serviceInven_result["body"]["startDate"] =
497         | DateTime.now().toUtc().toIso8601String();
498     serviceInven_result["body"]["endDate"] =
499         | DateTime.now().toUtc().toIso8601String();
500
501     CoreLog().info("new serviceInver : ${serviceInven_result["body"]}");
502
503     var new_serviceInven_result =
504         | await remoteDataSource.updateServiceInventoryNotification([
505             serviceInven_result["body"],
506             usecaseName: usecaseName,
507             usecaseStep: usecaseStep,
508         ]);
509     CoreLog().info(
510         | "nnew_serviceInven_result : ${new_serviceInven_result["body"]}");
511     return const Right(null);
512 } catch (e) {
513     return Left(ServerFailure(e.toString()));
514 }
515
516

```

## RemoteDataSource quiryServiceInventory

เมื่อรับ parameter มาแล้วจะทำการปั้น base haeder ,coreBaseOption สำหรับ queryProductInventory และทำการ requestNetwork ผ่าน socketHepler และส่ง response คืนกลับไป

```
197     @override
198     FuturequiryServiceInventory(Map<String, dynamic> serviceInventory,
199         {String? usecaseName, String? usecaseStep}) async {
200         Map<String, String> baseHeaders = {
201             "Accept": "application/json",
202             "Content-type": "application/json; charset=utf-8",
203         };
204         baseHeaders.addAll({
205             HeaderKey.topic.name: "mfaf.inquiryServiceInventory",
206             HeaderKey.orgService.name: "common.communication",
207             HeaderKey.tmfSpec.name: "TMF637",
208             HeaderKey.baseApiVersion.name: '4.0.0',
209             HeaderKey.eventResponse.name:
210                 "mfaf.serviceInventoryInquired,mfaf.inquiryServiceInventoryFailed",
211         });
212
213     try {
214         CoreBaseOption coreBaseOption = CoreBaseOption(
215             useCase: usecaseName,
216             useCaseStep: usecaseStep,
217             headers: baseHeaders,
218             body: serviceInventory,
219             timeout: const Duration(seconds: 60),
220         ); // CoreBaseOption
221         CoreLog().info(
222             '+-----quiryServiceInventory body request+++++++' + ${serviceInventory});
223
224         var response = await socketHelper.requestNetwork(coreBaseOption);
225
226         CoreLog().info(
227             '=====quiryServiceInventory response datasource===== ${response}');
228         return response;
229     } catch (e) {
230         rethrow;
231     }
232 }
233 }
```

## RemoteDataSource queryProductInventory

```
235  
236     @override  
237     Future<Map<String, dynamic>> updateServiceInventoryNotification(  
238         {String? usecaseName,  
239          String? usecaseStep}) async {  
240         Map<String, String> baseHeaders = {  
241             "Accept": "application/json",  
242             "Content-type": "application/json; charset=utf-8",  
243             "useCaseName": "",  
244             "useCaseStep": ""  
245         };  
246         baseHeaders.addAll({  
247             HeaderKey.topic.name: "mfaf.updateServiceInventoryNotification",  
248             HeaderKey.orgService.name: "common.communication",  
249             HeaderKey.tmfSpec.name: "TMF681",  
250             HeaderKey.baseApiVersion.name: '4.0.0',  
251             HeaderKey.eventResponse.name:  
252                 "mfaf.serviceInventoryNotificationUpdated,mfaf.updateServiceInventoryFailed",  
253         });  
254  
255         try {  
256             CoreBaseOption coreBaseOption = CoreBaseOption(  
257                 useCase: usecaseName,  
258                 useCaseStep: usecaseStep,  
259                 headers: baseHeaders,  
260                 body: serviceInventory,  
261             );  
262             CoreLog().info(  
263                 '|++++++updateServiceInventoryNotification body request+++++++$serviceInventory')  
264         } catch (e) {  
265             var response = await socketHelper.requestNetwork(coreBaseOption);  
266             CoreLog().info(  
267                 '|=====updateServiceInventoryNotification response datasource=====+$response');  
268             return response;  
269         } catch (e) {  
270             rethrow;  
271         }  
272     }  
273 }
```

## Push Notification

### การส่ง Push Notification

ส่งเข้าไปยัง user จะมี interface ของ tfm-681 communication โดยจะผ่าน api createNotiIndividual ซึ่งรายละเอียดทั้งหมดสามารถดูได้ที่เอกสาร T16\_MFAF\_Communication\_Interface\_Specification

โดยปกติทาง Tester ที่ทำการทดสอบจะใช้ tool ที่ชื่อว่า BIM (<https://staging-bim.ais.co.th/>) ที่เป็น web application เข้ามาช่วยในการส่ง ทำให้ไม่ต้องกำหนด parameter ต่างๆ ด้วยตัวเองและมีความสะดวก สบายการใช้งาน แต่สำหรับ develop จะมีช่องทางอื่นๆ ที่สามารถส่ง notification ได้ เช่น กัน อย่างเช่น การ produce event ไปยัง kafka ตรง ผ่าน kafka connect: confluent ที่หน้า consol และ การใช้ rest over kafka ซึ่งจะนำมาเป็นตัวอย่างต่อไปนี้

จาก script ตัวอย่าง จะทำการยิง curl api ไปยัง endpoint ที่มีการได้มีการเพิ่ม rest over kafka ไว้แล้ว ในที่นี้จะแนะนำการปรับ payload เป็นต้นเพื่อการทดสอบ

#### *filed subject , content*

ใน body จะมี ชึ้น 2 ตัวนี้จะกลายเป็น default Title , body ของ os notification เป็นองค์นี้ที่ไม่สามารถ handle ภาษาได้ โดย **content** จะ**ไม่สามารถใช้ชี้ได้** ในช่วงเวลาวันเดียวกัน

#### *localizedString*

จะเป็น array object ที่จะบรรจุตัว subject , content ในภาษาต่างๆ โดยจะถูกนำไปใช้ใน inbox และตัว os notification กรณีที่สามารถเข้ามาจัด ภาษาได้

#### *TORO\_notificationCategory*

เป็นตัวกำหนด category ของ notification ชึ่งค่าที่เป็นไปได้จะใช้ได้แค่ ตาม default\_category\_push\_notification ในไฟล์ constant

#### *receiver*

เป็น array object ที่จะบรรจุตัว appUserId ที่ระบุเบอร์ที่ต้องการส่ง

#### *attachment*

จะเป็น object ที่จะระบุ ส่งที่ต้องการแนบไปกับ notification เช่นรูป ลิงปล้ายทางที่ต้องการเปิด

```
curl --location
'https://pkc-4j8dq.southeastasia.azure.confluent.cloud:443/kafka/v3/clusters/lkc-p5d25k/topics/mfaf.createNotiIndividual/records' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic
NUFaVTZHNjdOUk1JRTM3Ujpiak9Ha09zV11xaFdHdW5ObEFETHk1emxzR2h6QkNEWGdzc1JFSTJJMG4vOWplVFE0dk02zzFpdW9MaG1tRWI1' \
--data '{
  "value": {
    "type": "JSON",
    "data": {
      "body": {
        "subject": "The package has been successfully cancelled.",
        "content": "To view additional add-on packages. 14:55:23",
        "localizedString": [
          {
            "locale": "th-TH",
            "attribute": "subject",
            "string": "แพ๊กเกจได้ถูกยกเลิกเรียบร้อยแล้ว"
          }
        ]
      }
    }
  }
}'
```

```
        } ,
        {
            "locale":"th-TH",
            "attribute":"content",
            "string":"คุณสามารถดูแพ็คเกจเสริมอื่นๆ ได้ที่เมนู \"ซื้อแพ็คเกจเสริม\""
        } ,
        {
            "locale":"en-US",
            "attribute":"subject",
            "string":"The package has been successfully cancelled."
        } ,
        {
            "locale":"en-US",
            "attribute":"content",
            "string":"You can view additional add-on packages in \"Buy on-top packages\" menu."
        }
    ] ,
    "messageType":"Push Notification by Individual",
    "priority":"high",
    "state":"inProgress",
    "sender":{
        "name":"ProductOrder"
    },
    "receiver":[
        {
            "appUserId":"0934000838"
        }
    ],
    "characteristic":[
        {
            "name":"appId",
            "valueType":"string",
            "value":"454400030025003"
        },
        {
            "name":"appName",
            "valueType":"string",
            "value":"MyAIS"
        },
        {
            "name":"appVersion",
            "valueType":"string",
            "value":"10.0.0"
        }
    ]
}
```

```
        }
    ],
    "attachment": [
        {
            "name": "actionLink",
            "url": "https://mws.adldigitalservice.com/openWebToApp?URI=myais%3A%2F%2FCurrentPa
ck"
        }
    ],
    "TORO_notificationType": "individual",
    "TORO_notificationCategory": "promotion",
    "TORO_receiverType": "asset",
    "TORO_pushNotification": true
},
"header": {
    "baseApiVersion": "4.0.0",
    "broker": "",
    "channel": "SyncData",
    "communication": "unicast",
    "from": "customer.product-order",
    "groupTags": [
        ...
    ],
    "identity": {
        ...
    },
    "orgService": "CDC",
    "schemaVersion": "1",
    "session": "c746bdc5-467f-4a31-b2d6-3e179c61d67a",
    "timestamp": "2024-10-28T09:37:23.091Z",
    "tmfSpec": "TMF622",
    "transaction": "21973704-c5b6-4152-a42d-c2299044f7df",
    "useCase": "promotionOrder",
    "useCaseAge": 2,
    "useCaseStep": "0",
    "version": "2.0",
}

"instanceData": "eyJhZGRpdGlvbmFsSW5mb051bWJlcI16Ik1qQTVNemN3TlRVMU1EVWkifQ=="
}
}
}
}'
```

## การแสดง notification foreground

การแสดง notification foreground เป็นการแสดง notification ขณะที่ user กำลังใช้งาน app อยู่ การแสดง notification foreground จะใช้งาน service สອตัวนั้นคือ push notification service ร่วมกับ local\_notification\_helper ในการทำงาน

### onMessage

ขณะที่ app จะเปิดใช้งาน push notification service onMessage ที่จะทำการ listen FirebaseMessaging.onMessage ที่จะทำการส่ง notification จากเครื่องเข้ามาที่ app หลังจากนั้นจะทำการเช็คว่า notification ที่ถูกส่งมาเป็นของเก่าหรือไม่ ถ้าไม่จะทำการเรียก LocalNotificationHelper.showNotificationOnChanel(message) เพื่อแสดงต่อไป

```
356     void onMessage() {
357         FirebaseMessaging.onMessage.listen(
358             (RemoteMessage message) {
359                 CoreLog()
360                     .info("Receive FirebaseMessaging onMessage: ${message.toString()}");
361                 if (!onMessageIdHistory.contains(message.messageId.toString())) {
362                     CoreLog().info("Message is Unique: ${message.messageId.toString()}");
363                     onMessageIdHistory.add(message.messageId.toString());
364                     LocalNotificationHelper.showNotificationOnChanel(message);
365                     setMoreIconAppBadge();
366                 } else {
367                     CoreLog()
368                         .info("Message is Duplicate: ${message.messageId.toString()}");
369                 }
370             },
371         );
372     }
373 }
```

### LocalNotificationHelper showNotificationOnChanel

เมื่อเริ่มทำงานจะทำการ decode notification data ที่แนบเข้าเพื่อทำการเช็ค category ใน setting ผ่าน getCatAlertAllowed ถ้าหากเป็น category ที่เปิดไว้จะทำการเรียก buildDetailAndShowNotification เพื่อแสดงต่อไป

```

161 static Future<void> showNotificationOnChanel(RemoteMessage message,
162   | {bool? onBackground}) async {
163   //LocalNotificationHelper.initialize();
164   //final notification = message.notification;
165
166   final NotiData = jsonEncode(message.data);
167   final NotiDataJson = jsonDecode(NotiData);
168
169   final contentDetail = jsonEncode(NotiDataJson["contentDetail"]);
170   final contentDetailJson = jsonDecode(jsonDecode(contentDetail));
171
172   final onNotiCategory = contentDetailJson["T0R0_notificationCategory"];
173
174   if (await getCatAlertAllowed(onNotiCategory)) {
175     //CoreLog().info("Do Alert This Noti, Category: $onNotiCategory ");
176     CoreLog().info("Do Alert This Noti, Category: $onNotiCategory ");
177     //CoreLog().error(message);
178     bool isBackground = onBackground ?? false;
179     CoreLog().info("isBackground : $isBackground");
180     await buildDetailAndShowNotification(
181       message: message,
182       isBackground: isBackground,
183     );
184   } else {
185     CoreLog().info("User not Allowed Categoty on This MSG");
186   }
187
188   final onRemoteCategoryForDev =
189     contentDetailJson["Remote_Notification_Category_For_Dev"];
190
191   if (onRemoteCategoryForDev != null) {
192     setRemoteCategoryForDev(onRemoteCategoryForDev);
193   }
194

```

### *getCatAlertAllowed*

จะทำการเช็คว่าcategory ที่แนบมา กับ message มีการ allowed และถูกเก็บไว้ใน SharedPreferences หรือไม่

```

235 static Future<bool> getCatAlertAllowed(String onNotiCategory) async {
236   bool result = true;
237   final prefs = await SharedPreferences.getInstance();
238   try {
239     var on_prefs = prefs.getStringList(prefs_key_allowedNotificationCategory);
240
241     var for_dev = prefs.getStringList(prefs_key_allowedNotiCategoryForDev);
242     if (for_dev != null && for_dev.isNotEmpty) {
243       on_prefs = on_prefs! + for_dev;
244     }
245
246     if (on_prefs != null) {
247       // CoreLog().info(
248       //   "on getCatAlertAllowed on TopicMSG: $onNotiCategory, current pref: ${prefs
249       result = on_prefs.contains(onNotiCategory);
250     } else {
251       result = false;
252     }
253   } catch (e) {
254     result = false;
255   }
256   return result;
257 }

```

### *buildDetailAndShowNotification*

เมื่อเริ่มทำงาน จะทำการสร้าง notiMap ที่ได้จาก message .data และทำการเพิ่ม title, body ของ message เข้าไป หลังจากนั้น decode ให้เป็น json  
สร้าง androidNotiDetail สำหรับแสดงบน android , darwinNotificationDetails สำหรับแสดงบน ios

ทำการสร้าง noti\_title มีค่า default = myAIS , noti\_detail มีค่า default = New Notification  
จากนั้นทำการเช็ค notification.body และดึงค่าจาก message.notification.title c] และ message.notification.body มาใช้งานแทนค่า default

```

274     static Future<void> buildDetailAndShowNotification(
275         required RemoteMessage message, required bool isBackground) async {
276         Map<String, dynamic> notiMap = message.data;
277         notiMap.addAll({
278             "title": message.notification?.title,
279             "body": message.notification?.body,
280         });
281
282         final NotiData = jsonEncode(notiMap);
283         final NotiDataJson = jsonDecode(NotiData);
284
285         var androidNotiDetail;
286         var darwinNotificationDetails;
287
288         int noti_id = message.messageId.hashCode; //message.notification.hashCode;
289         String noti_title = "myAIS";
290         String noti_detail = "New Notification"; //New Notification";
291
292         //set noti title / detail with notification payload
293         if (message.notification != null) {
294             if (message.notification!.body != null) {
295                 noti_title = message.notification!.title!;
296                 noti_detail = message.notification!.body!;
297             }
298         }

```

หลังจากนั้นทำการเช็คภาษาที่ใช้งานอยู่โดยจะเช็คผ่าน Core CurrentLang().currentLanguage ถ้ามีการรับ notification จาก background (ปัจจุบันยกเลิกแล้ว) จะเช็คผ่าน SharedPreferences แทน

```

299
300         //set noti title / detail with localized payload
301         var currentLang = "*";
302         try {
303             if (!isBackground) {
304                 CurrentLang();
305                 currentLang = CurrentLang().currentLanguage;
306                 CoreLog().info("select lang foreground: $currentLang");
307             } else {
308                 final prefs = await SharedPreferences.getInstance();
309                 currentLang = prefs.getString(prefs_key_notification_lang) ?? "en";
310                 CoreLog().info("select lang isBackground: $currentLang");
311             }
312         } catch (exception) {
313             try {
314                 final prefs = await SharedPreferences.getInstance();
315                 currentLang = prefs.getString(prefs_key_notification_lang) ?? "en";
316                 CoreLog().info("select lang prefs: $currentLang");
317             } catch (exception) {
318                 CoreLog().info("select lang prefs exception: ${exception.toString()}");
319                 currentLang = "en";
320             }
321         }

```

หลังจากได้ภาษาที่ต้องการแสดงแล้ว จะทำการตั้ง localizedString ที่อยู่ใน contentDetail ของ message โดยจะเริ่มการ Decode ให้เป็น json ทำการวนลูปภายใต้ localizedString ที่มีการระบุภาษา ตรงกับภาษาที่จะใช้ในการแสดง

```

322     try {
323         final contentDetail = jsonEncode(NotiDataJson["contentDetail"]);
324         final contentDetailJson = jsonDecode(jsonDecode(contentDetail));
325
326         if (contentDetailJson["localizedString"] != null) {
327             var localizedStrings = contentDetailJson["localizedString"];
328
329             int i = 0;
330             for (i; i < localizedStrings.length; i++) {
331                 var localizedString = localizedStrings[i];
332                 CoreLog().info(localizedString);
333                 if (localizedString["locale"].toString().contains(currentLang)) {
334                     if (localizedString["attribute"].toString().toLowerCase() ==
335                         "content" ||
336                         localizedString["attribute"].toString().toLowerCase() ==
337                         "detail") {
338                         noti_detail = localizedString["string"];
339                     }
340                     if (localizedString["attribute"].toString().toLowerCase() ==
341                         "subject" ||
342                         localizedString["attribute"].toString().toLowerCase() ==
343                         "title") {
344                         noti_title = localizedString["string"];
345                     }
346                 }
347             }
348         }
349     } catch (exception) {
350         CoreLog().info(
351             "set noti title / detail with localized payload error: ${exception.toString()}");
352     }
353

```

จากนั้นทำการ decode dataMessage และ เช็ค mediaDataList เพื่อตึง mediaUrl ที่เป็นลิงค์รูปภาพอุปกรณ์ โดยเลือกเฉพาะ mediaNo 1 เท่านั้น

```

354     String mediaUrl = "";
355     //String mediaType = "";
356     String mediaExtension = "";
357     try {
358         final dataMessage = jsonEncode(NotiDataJson["dataMessage"]);
359         final dataMessageString = jsonDecode(dataMessage);
360         final dataMessageJson = jsonDecode(dataMessageString);
361         List<dynamic> mediaDataList = dataMessageJson["mediaDataList"];
362
363         if (mediaDataList.isNotEmpty) {
364             int i = 0;
365             for (i; i < mediaDataList.length; i++) {
366                 if (mediaDataList[i]["mediaNo"].toString() == "1") {
367                     mediaUrl = mediaDataList[i]["mediaUrl"];
368                     //mediaType = mediaDataList[i]["mediaType"];
369                     mediaExtension = mediaDataList[i]["mediaExtension"];
370                 }
371             }
372         }
373     } catch (exception) {
374         CoreLog().info(
375             "set noti media with mediaDataList payload error: ${exception.toString()}");
376     }
377

```

ถัดมาจะทำการเช็ค platform เพื่อแสดง notification

## Android Platform

ถ้าหากมีรูปที่ต้องการแสดงจะทำการ ดึงรูปแล้วแปลงเป็น byte ด้วย `_getByteArrayFromUrl` และทำการ set `AndroidNotificationDetails` และทำการสั่ง `show`

```
77  
378     if (Platform.isAndroid) {  
379         if (mediaUrl.isNotEmpty) {  
380             try {  
381                 final ByteArrayAndroidBitmap bigPicture =  
382                     ByteArrayListAndroidBitmap(await _getByteArrayFromUrl(mediaUrl));  
383  
384                 final BigPictureStyleInformation bigPictureStyleInformation =  
385                     BigPictureStyleInformation(  
386                         bigPicture,  
387                         largeIcon: bigPicture,  
388                     );  
389                 androidNotiDetail = AndroidNotificationDetails(  
390                     androidNotificationChannel.id,  
391                     androidNotificationChannel.name,  
392                     channelDescription: androidNotificationChannel.description,  
393                     styleInformation: bigPictureStyleInformation,  
394                 );  
395             } catch (e) {  
396                 androidNotiDetail = AndroidNotificationDetails(  
397                     androidNotificationChannel.id,  
398                     androidNotificationChannel.name,  
399                     channelDescription: androidNotificationChannel.description,  
400                 );  
401             }  
402         } else {  
403             androidNotiDetail = AndroidNotificationDetails(  
404                 androidNotificationChannel.id,  
405                 androidNotificationChannel.name,  
406                 channelDescription: androidNotificationChannel.description,  
407             );  
408         }  
409         CoreLog().info(  
410             "Alert Noti Android: id:$noti_id, notiTitle:$noti_title, notiDetail: $noti_detail, mediaUrl: $mediaUrl");  
411         await NOTI_PLUGIN.show(  
412             noti_id,  
413             noti_title,  
414             noti_detail,  
415             payload: NotiData,  
416             NotificationDetails(android: androidNotiDetail),  
417         );  
418     }  
419 }
```

## iOS Platform

ถ้าหากมีรูปที่ต้องการแสดงจะทำการ ดึงรูปแล้ว `save` ด้วย `_downloadAndSaveFile` และทำการ set `DarwinNotificationDetails` และทำการสั่ง `show`

```
420     if (Platform.isIOS) {  
421         if (mediaUrl.isNotEmpty && !isBackground) {  
422             try {  
423                 final String bigPicturePath = await _downloadAndSaveFile(  
424                     mediaUrl,  
425                     "$noti_id.$mediaExtension",  
426                 );  
427                 darwinNotificationDetails = DarwinNotificationDetails(  
428                     attachments: <DarwinNotificationAttachment>[  
429                         DarwinNotificationAttachment(  
430                             bigPicturePath,  
431                             hideThumbnail: false,  
432                         ) // DarwinNotificationAttachment  
433                     ], // <DarwinNotificationAttachment>[]  
434                 ); // DarwinNotificationDetails  
435             } catch (e) {  
436                 darwinNotificationDetails = const DarwinNotificationDetails();  
437             }  
438         } else {  
439             darwinNotificationDetails = const DarwinNotificationDetails();  
440         }  
441         CoreLog().info(  
442             "Alert Noti IOS: id:$noti_id, notiTitle:$noti_title, notiDetail: $noti_detail, mediaUrl: $mediaUrl");  
443         await NOTI_PLUGIN.show(  
444             noti_id,  
445             noti_title,  
446             noti_detail,  
447             payload: NotiData,  
448             NotificationDetails(ios: darwinNotificationDetails),  
449         );  
450     }  
451 }
```

## Notification Service

## FCM TOKEN

fcm token เป็น token เพื่อใช้ในการระบุตัวเครื่องที่ต้องการส่ง push notifications

### get fcm token

จะเรียกผ่าน FirebaseMessaging.instance.getToken โดย ใน pushNotificationService จะมี method ที่เรียกใช้งานตามด้านล่าง โดยหลังจากได้รับtoken จะทำการเรียกใช้งาน method setToken

```
433 void getTokenFromFCM() async {
434   CoreLog().info("on getTokenFromFCM");
435   FirebaseMessaging.instance.getToken().then(setToken);
436 }
437
438 void refreshTokenFromFCM() async {
439   CoreLog().info("on refreshTokenFromFCM");
440   FirebaseMessaging.instance.onTokenRefresh.listen(setToken);
441 }
442
443 void deleteTokenFromFCM() async {
444   await FirebaseMessaging.instance.deleteToken();
445 }
446
```

### setToken

จะรับ token แล้วนำไปวางที่ realm core ผ่าน CoreUtils().setFCMTOKEN()

```
374 Future<void> setToken(String? token) async {
375   CoreLog().info('on setToken => FCM Token: $token');
376   FCM_TOKEN = token!;
377
378   CoreLog().info("FCMToken: $FCM_TOKEN");
379   await CoreUtils().setFCMTOKEN(tokenName: "fcmToken", tokenValue: FCM_TOKEN);
380   reSubscribeTopicToFCM();
381 }
```

### reSubscribeTopicToFCM

ดึงค่า setting จาก SharedPreferences และส่งไป subscriToTopicFCM

```
471 Future<void> |reSubscribeTopicToFCM() async {
472   try {
473     final prefs = await SharedPreferences.getInstance();
474     var allowedNotificationCategory =
475       prefs.getStringList(prefs_key_allowedNotificationCategory);
476     if (allowedNotificationCategory != null) {
477       for (int i = 0; i < allowedNotificationCategory.length; i++) +
478         subscribeToTopicFCM(allowedNotificationCategory[i]);
479     }
480   }
481 } catch (e, s) {
482   CoreLog().e Type: String,
483   "Exception on reSubscribeTopicToFCM",
484   error: e,
485   stackTrace: stack,
486 };
487 }
488 }
```

## FCM Topic Subscription

การ subscribe และ un subscribe เป็นการจัดการของการส่ง broadcast push notification

โดยมี method สำหรับจัดการใน push notification service

```
489
490     Future<void> subscribeToTopicFCM(String topic) async {
491         CoreLog().info("on subscribe to FCM Topic $topic");
492         await FirebaseMessaging.instance.subscribeToTopic(topic);
493     }
494
495     Future<void> unsubscribeFromTopicFCM(String topic) async {
496         CoreLog().info("on unsubscribe to FCM Topic $topic");
497         await FirebaseMessaging.instance.unsubscribeFromTopic(topic);
498     }
499 
```

## CommunicationListenerEventService

ใช้สำหรับจัดการ core event, stream event ที่เกิดขึ้นภายใน app ที่ communication จะเป็นต้องจัดการร่วมใน event นั้นๆ

## การเริ่มต้นทำงาน

จะเริ่มต้นการทำงานผ่าน initializeCommunicationListenerEventService โดยจะทำการเริ่ม subscribe Core event หลัก 3 ตัว รวมทั้ง stream

1 ISLOGIN สำหรับรับ event เกี่ยวกับการ login app เมื่อมี event จะทำการเรียก loginEventHandler

2 currentLanguageChanged สำหรับรับ event การเปลี่ยนภาษาของ app เมื่อมี event จะทำการเรียก currentLanguageChangeEventHandle

3 เริ่มทำการ stream alreadyNavigateFlagStream เพื่อใช้ทำงานใน oda

4 event\_type\_homeIsReady สำหรับรับ event เมื่อหน้า app home โหลดเสร็จแล้วเมื่อมี event จะทำการเรียก setflagFromHome

```
44 |     initializeCommunicationListenerEventService() {
45 |         //event log in
46 |         if (!CoreEvent().isAlreadySubscriptionEvent(
47 |             type: 'ISLOGIN', callback: loginEventHandler)) {
48 |             CoreLog().debug('@@event ISLOGIN On Communication is Register');
49 |             CoreEvent().on('ISLOGIN', loginEventHandler);
50 |         }
51 |         //event setting language
52 |         if (!CoreEvent().isAlreadySubscriptionEvent(
53 |             type: 'currentLanguageChanged',
54 |             callback: currentLanguageChangeEventHandle)) {
55 |             CoreLog().debug(
56 |                 '@@event currentLanguageChanged On Communication is Register');
57 |             CoreEvent()
58 |                 .on('currentLanguageChanged', currentLanguageChangeEventHandle);
59 |         }
60 |
61 |         //get flag from home
62 |         alreadyNavigateFlagStream.stream;
63 |         if (!CoreEvent().isAlreadySubscriptionEvent(
64 |             type: event_type_homeIsReady, callback: setflagFromHome)) {
65 |             CoreLog().debug('@@event $event_type_homeIsReady is Register');
66 |             CoreEvent().on(event_type_homeIsReady, setflagFromHome);
67 |         }
68 |     }
```

## loginEventHandler

การทำงานของ loginEventHandler หลังเริ่มการทำงานจะทำการ delay 5 วินาทีเพื่อให้ข้อมูลบน realm ที่จำเป็นมีการ sync ให้สมบูรณ์ก่อนหลังจากนั้นจะทำการเรียก listenerCommunicationRealm ที่เป็น event listener เมื่อมีการข้อมูล update ข้อมูลบน realm หลังจากนั้นจะทำการเช็คว่า มีการ login และ

หรือไม่ เพราะ **ISLOGIN** event จะเกิดขึ้นทั้งตอน login และ logout ถัดมาเมื่อเช็คแล้วว่ามีการ login จะทำการเรียก **getAllowedNotificationCategory**

```
120 |     dynamic loginEventHandle(value) async {
121 |       try {
122 |         CoreLog().info("got Event loginEventHandle: $value");
123 |         //Waiting 5 sec for realm stable after sync
124 |         await Future.delayed(const Duration(seconds: 5));
125 |         //re StreamSubscribe realm
126 |         listenerCommunicationRealm();
127 |         if (await isLogin()) {
128 |           //Sync setting notification and re-subscribe fcm topic
129 |           String serviceInventID = "";
130 |           List<String> remoteAllowedCategoryNotification = [];
131 |           var result = await viewCommunicationMessagesUseCase
132 |             .getAllowedNotificationCategory(await getMyid());
133 |         }
134 |       }
135 |     }
136 |   }
137 | }
```

จากนั้นจะทำการดึง **result** ที่ได้ออกมาแล้วเช็คว่าไม่ค่าว่างหรือไม่ ถ้าว่างจะทำการใช้ค่า **setting** จาก **SharedPreferences** หรือถ้าใน **SharedPreferences** ว่างจะใช้ค่า **default** แทน จากนั้นทำการ **force** เปิด bill category จากนั้นทำการลบ ภาษาที่อยู่ใน category ออก และทำการ เรียก **setAllowedNotificationCategory** เพื่อบันทึก **setting**

```
134 |     result.fold(
135 |       (l) {},
136 |       (r) {
137 |         serviceInventID = r["serviceInvenID"];
138 |         remoteAllowedCategoryNotification = r["category"];
139 |       },
140 |     );
141 |     final prefs = await SharedPreferences.getInstance();
142 |     if (remoteAllowedCategoryNotification.isEmpty) {
143 |       remoteAllowedCategoryNotification =
144 |         prefs.getStringList(prefs_key_allowedNotificationCategory) ??
145 |           defult_category_push_notifiaction;
146 |     }
147 |     //Force add bill category
148 |     if (!remoteAllowedCategoryNotification
149 |         .contains(bill_category_push_notifiaction)) {
150 |       remoteAllowedCategoryNotification =
151 |         .add(bill_category_push_notifiaction);
152 |     }
153 |
154 |     //Delete all lang in current category
155 |     remoteAllowedCategoryNotification.removeWhere(
156 |       (category) => CurrentLang().localeList.contains(category));
157 |
158 |     prefs.setString(
159 |       prefs_key_notification_lang, CurrentLang().currentLanguage);
160 |
161 |     viewCommunicationMessagesUseCase.setAllowedNotificationCategory(
162 |       category: remoteAllowedCategoryNotification,
163 |       serviceInventID: serviceInventID);
164 |   }
165 | }
```

กรณีที่เช็คแล้วไม่พบการ login จะมองเป็นการ logout จะทำการ clear notification ของ user ก่อนหน้าที่ค้าง notification bar ของเครื่องออก

```
164 |   } else {
165 |     CoreLog().info("Try to clearAllNotifications");
166 |     LocalNotificationHelper.NOTI_PLUGIN.cancelAll();
167 |     Eraser.clearAllAppNotifications();
168 |   }
169 | } catch (e, _) {
170 |   //
171 |   CoreLog().error(
172 |     "loginEventHandle has Exception: ${e.toString()}",
173 |     error: e,
174 |     stackTrace: _,
175 |   );
176 | }
177 | }
```

currentLanguageChangeEventHandle

การทำงานจะคล้ายกับ Event login เช่นมา แต่จะทำงานเฉพาะการ setAllowedNotificationCategory จากค่าเดิมเพื่อให้ทำการปรับค่า ภาษาใน setting

```
179 dynamic currentLanguageChangeEventHandle(value) async {
180   CoreLog().info("got Event From currentLanguageChangeEventHandle: $value");
181
182   final prefs = await SharedPreferences.getInstance();
183   String? prefs_notification_lang =
184     prefs.getString(prefs_key_notification_lang);
185
186   if (prefs_notification_lang != null &&
187     prefs_notification_lang != value.toString()) {
188     String serviceInvenID = "";
189     List<String>? category =
190       prefs.getStringList(prefs_key_allowedNotificationCategory);
191     var result = await viewCommunicationMessagesUseCase
192       .getAllowedNotificationCategory(await getMyid());
193
194     result.fold(
195       (l) {},
196       (r) {
197         category = r["category"];
198         serviceInvenID = r["serviceInvenID"];
199       },
200     );
201     // List<String>? category =
202     //   prefs.getStringList(prefs_key_allowedNotificationCategory);
203     viewCommunicationMessagesUseCase.setAllowedNotificationCategory(
204       category: category ?? ["-"], serviceInvenID: serviceInvenID);
205   }
206 }
```

## setflagFromHome

จะทำการ set flag พร้อมกับ add event ไปที่ flagHomeAppAlraday เพื่อนำไปใช้งาน จากนั้นทำการปิด core event ของ event\_type\_homeIsReady และทำการเรียก listenerCommunicationRealm เพื่อเริ่มรับ event จาก realm ในมี และ CommunicationRepositoryImpl clearTempReferntMessage เพื่อเคลียร์ค่า message temp refferent

```
208 dynamic setflagFromHome(value) {
209   CoreLog().info("got Event From Home: $value");
210   flagHomeAppAlraday = true;
211   alraedyNavigateFlagStream.add('Event emitted');
212   CoreEvent().off(type: event_type_homeIsReady);
213   listenerCommunicationRealm();
214   communicationRepositoryImpl.clearTempReferntMessage(delayMillsec: 500);
215 }
```

## listenerCommunicationRealm

listenerCommunicationRealm ทำหน้าที่เริ่มต้นการทำงานของ event listener เมื่อมีข้อมูล update ข้อมูลบน realm communication ทั้ง 3 ตัวได้แก่ CommunicationMessageMaster, CommunicationMessage, CommunicationMessageAnonymous โดยจะทำการเรียก localDataSource เพื่อนำมา RealmResults ที่มีความสามารถในการ stream event ในการเรียกสร้าง StreamSubscription เพื่อ handle ดังนี้

streamSubscriptionCommunicationMessageMaster เป็น StreamSubscription เมื่อมีข้อมูลถูกอัปเดตที่ CommunicationMessageMaster โดยจะทำการเรียกใช้งาน getBadgeAndEmitCoreEvent

2 streamSubscriptionCommunicationMessage เป็น StreamSubscription เมื่อมีข้อมูลถูกอัปเดตที่ CommunicationMessage โดยจะทำการเรียกใช้งาน getBadgeAndEmitCoreEvent

3 communicartionRealmResults จะทำการ listener CommunicationMessageAnonymous ด้วยทำการเรียกใช้งาน checkEventAndClearTempReferentMessage

```
80    listenerCommunicationRealm() async {
81      RealmResults<CommunicationMessageMaster> communicationMasterRealmResults =
82        await communicationRepositoryImpl.localDataSource
83          .getSteamCommunicationMessagesMaster();
84
85      RealmResults<CommunicationMessage> communicationRealmResults =
86        await communicationRepositoryImpl.localDataSource
87          .getSteamCommunicationMessages();
88
89      RealmResults<CommunicationMessageAnonymous> communicartionRealmResults =
90        await communicationRepositoryImpl
91          .communicationMessageAnonymousRepositoryImplForRefTemp
92          .anonymousLocalDataSourceImpl
93          .getCommunicationMessagesAnonymous();
94
95      streamSubscriptionCommunicationMessageMaster =
96        communicationMasterRealmResults.changes.listen(event) async {
97          getBadgeAndEmitCoreEvent();
98        };
99
100     streamSubscriptionCommunicationMessage =
101       communicationRealmResults.changes.listen(event) async {
102         getBadgeAndEmitCoreEvent();
103       };
104
105     communicartionRealmResults.changes.listen(event) async {
106       checkEventAndClearTempReferentMessage(event);
107     };
108   }
```

## getBadgeAndEmitCoreEvent

เป็น method ที่จะทำการเรียก usecase getBadgesUnReadMessages และทำการ emit จำนวน badge ผ่าน core event notiCount

```
217     getBadgeAndEmitCoreEvent() async {
218         var result =
219             await viewCommunicationMessagesUseCase.getBadgesUnReadMessages();
220         result.fold((l) {}, (r) {
221             CoreEvent().emit("notiCount", r);
222         });
223     }
```

## checkEventAndClearTempReferentMessage

checkEventAndClearTempReferentMessage เป็น method ที่จะรับ event จากการอัปเดตข้อมูลต่อ CommunicationMessageAnonymous ที่มาจากการอ่าน-ลง ข้อความของ user โดยจะรับ parameter event เข้า หลังจากนั้นจะทำการเช็คว่า login อยู่หรือไม่จากนั้นทำการ stamp เวลาที่มี event หลังจากนั้นรอ 1 วินาที ถ้าภายใน 1 วินาทีไม่มี event เข้ามาใหม่ จะทำการเรียก CommunicationRepositoryImpl clearTempReferntMessage เพื่อเคลียร์ค่า message temp referent

```
225     checkEventAndClearTempReferentMessage(
226         RealmResultsChanges<CommunicationMessageAnonymous> event) async {
227         bool isLoggedIn = await isLogin();
228         if (isLoggedIn && event.inserted.isNotEmpty) {
229             var tempDateTimeClearEven = DateTime.now();
230             lastDateTimeClearEvent = tempDateTimeClearEven;
231             // wait 1 sec if no another event change lastDateTimeClearEvent, it's will match and let clear temp
232             await Future.delayed(const Duration(seconds: 1));
233             if (lastDateTimeClearEvent == tempDateTimeClearEven) {
234                 int delayMillsec = 250;
235                 communicationRepositoryImpl.clearTempReferntMessage(
236                     delayMillsec: delayMillsec,
237                 );
238             }
239         }
240     }
```

## clearTempReferntMessage

clearTempReferntMessage เป็น service สำหรับเคลียร์ค่า read- delete ที่ยังไม่ถูกส่งไปยัง DOS เป็น service ที่เข้าไปปัดการกับ remote data source รวมอยู่ใน CommunicationRepositoryImpl การทำงานจะดึงข้อมูล allRefTempMessages จาก CommunicationMessageAnonymous ทั้งหมด ขึ้นมาแล้วทำการ วนลูปตามข้อมูลที่ได้รับ ภายในลูปจะทำการเช็คว่า มีการส่ง parameter delayMillsec มาให้หรือไม่ เพื่อใช้ในการตั้ง delay สำหรับการจำกัด transaction ที่จะถูกยิงออกไป

```
983     Future<void> clearTempReferntMessage({int? delayMillsec}) async {
984         try {
985             List<CommunicationMessageAnonymous> allRefTempMessages =
986                 await communicationMessageAnonymousRepositoryImplForRefTemp
987                     .anonymousLocalDataSourceImpl
988                     .getCommunicationMessagesAnonymous()
989                     .then((value) => value.toList());
990
991             String createdUserDatetime = DateFormat('yyyy-MM-dd@HH:mm:ss:ms', 'en')
992                 .format(await getOldestCreateDateUser());
993
994             int idxx = 0;
995             String myid = await getMyid();
996             for (var refTemp in allRefTempMessages) {
997                 if (delayMillsec != null && delayMillsec > 0) {
998                     await Future.delayed(Duration(milliseconds: delayMillsec));
999                 }
}
```

จากนั้น จะทำการ query CommunicationMessage หรือ ถ้าไม่มี result จะไป query CommunicationMessageMaster แทน โดยหากได้ result กลับมาจะทำการเทียบว่า state ทั้งสองที่เท่ากันหรือไม่ ถ้าหากเท่ากันจะทำการลบ RefTempMessages ตอนนั้น แต่ถ้าไม่เท่ากันจะทำการเช็คค่า state จาก RefTempMessages

กรณี state = read จะเรียก setFullRemoteReadedMessagesByID

กรณีอื่น (state = cancelled ) จะเรียก deleteFullRemoteCommunicationMessagesByID

```
1000     try {
1001         List<CommunicationMessage> noti = await localDataSource
1002             .queryCommunicationMessages("TORO_messageId = '${refTemp.id}' ")
1003             .then((value) => value.toList());
1004         if (noti.isNotEmpty) {
1005             if (refTemp.state != noti[0].state) {
1006                 if (refTemp.state == "read") {
1007                     await setFullRemoteReadedMessagesByID(
1008                         msg_id: noti[0].id,
1009                         myid: myid,
1010                     );
1011                 } else {
1012                     await deleteFullRemoteCommunicationMessagesByID(
1013                         msg_id: noti[0].id,
1014                         myid: myid,
1015                     );
1016                 }
1017             } else {
1018                 communicationMessageAnonymousRepositoryImplForRefTemp
1019                     .anonymousLocalDataSourceImpl
1020                     .deleteCommunicationMessageAnonymousByID(refTemp.id);
1021             }
1022         } else [
1023             ] else {
1024                 List<CommunicationMessageMaster> notiMaster = await localDataSource
1025                     .queryCommunicationMessagesMaster(
1026                         "TORO_messageId = '${refTemp.id}' "
1027                         .then((value) => value.toList());
1028             if (notiMaster.isNotEmpty) {
1029                 if (notiMaster[0].id == "") {
1030                     if (refTemp.state == "read") {
1031                         await setFullRemoteReadedMessagesByID(
1032                             msg_id: notiMaster[0].id,
1033                             myid: myid,
1034                         );
1035                     } else {
1036                         await deleteFullRemoteCommunicationMessagesByID(
1037                             msg_id: notiMaster[0].id,
1038                             myid: myid,
1039                         );
1040                     }
1041                 } else {
1042                     communicationMessageAnonymousRepositoryImplForRefTemp
1043                         .anonymousLocalDataSourceImpl
1044                         .deleteCommunicationMessageAnonymousByID(refTemp.id);
1045                 }
1046             }
1047         }
1048     }
1049 }
```

## setFullRemoteReadedMessagesByID

setFullRemoteReadedMessagesByID จะทำการบัน payload สำหรับการ update state ของ message ให้เป็น read และเรียก RemoteDataSource updateReadState โดยจะทำการ query data จาก getCommunicationMessagesByID ถ้าหากมี result จะทำการเช็ค state ว่าเป็น read ไปหรือยัง และเป็น type individual ด้วยหรือไม่ กรณีไม่มี result จาก getCommunicationMessagesByID จะเรียก getCommunicationMessagesMasterByID แทนแล้วทำการเรียก updateReadState เมื่อมั่นคง

```
672 Future<Either<Failure, void>> setFullRemoteReadedMessagesByID(
673     @required String msg_id,
674     @required String myid,
675     String? usecaseName,
676     String? usecaseStep) async {
677     try {
678         CoreLog().info("setReadedMessagesByID ID: $msg_id");
679         var noti = null;
680         try {
681             noti = await localDataSource.getCommunicationMessagesByID(msg_id);
682             if (noti != null &&
683                 noti.state!.toLowerCase() != "read" &&
684                 noti.TORO_notificationType!.toLowerCase() == "individual") {
685                 CoreLog().info(
686                     "Noti DATA madatory: correlationId: ${noti.correlationId} ,TORO_messageId: ${noti.TORO_messageId}, TORO_NotificationType: ${noti.TORO_notificationType}");
687                 var setReadNoti = {
688                     "correlationId": [myid], //noti.correlationId,
689                     "messageType": "Mobile app push notification",
690                     "communicationMessage": [
691                         {
692                             "TORO_messageId": noti.TORO_messageId,
693                             "TORO_notificationType":
694                                 ||| noti.TORO_notificationType //individual"
695                         },
696                     ],
697                     "receiver": [
698                         {
699                             "appUserId": myid,
700                         } //noti.correlationId
701                     ],
702                     "state": "read",
703                     "sender": {"name": noti.sender.name},
704                     "lastModified": DateTime.now()
705                         .toUtc()
706                         .toIso8601String() //"2023-07-05T16:30:04.012Z"
707                 };
708                 var result = await remoteDataSource.updateReadState(
709                     setReadNoti,
710                     usecaseName,
711                     usecaseStep,
712                 );
713                 CoreLog().info("setReadedMessagesByID Result : $result");
714             }
715         }
716     }
717     try {
718         noti = await localDataSource.getCommunicationMessagesMasterByID(msg_id);
719         if (noti != null) {
720             var setReadNoti = {
721                 "correlationId": [myid], //noti.correlationId,
722                 "messageType": "Mobile app push notification",
723                 "communicationMessage": [
724                     {
725                         "TORO_messageId": noti.TORO_messageId,
726                         "TORO_notificationType":
727                             ||| noti.TORO_notificationType, //broadcast",
728                         "TORO_templateMessage": {
729                             "id": noti.id,
730                             "@referredType": "communicationMessageMaster"
731                         }
732                     },
733                 ],
734                 "receiver": [
735                     {
736                         "appUserId": myid,
737                         } //noti.correlationId
738                 ],
739                 "state": "read",
740                 "sender": {"name": noti.sender.name},
741                 "lastModified": DateTime.now()
742                     .toUtc()
743                     .toIso8601String() //"2023-07-05T16:30:04.012Z"
744             };
745             var result = await remoteDataSource.updateReadState(
746                 setReadNoti,
747                 usecaseName,
748                 usecaseStep,
749             );
750             CoreLog().info("setReadedMessagesByID Result : $result");
751         }
752     } catch (e) {
753         // 
754     }
755 
```

## deleteFullRemoteCommunicationMessagesByID

deleteFullRemoteCommunicationMessagesByID จะทำการปั้น payload สำหรับการ update state ให้เป็น cancelled หรือ update deleteDate ของ message และเรียก RemoteDataSource updateDeleteState ถ้าหากมี result จากทำการ query data จาก get Communication MessagesByID และจะทำการ ลบ message นั้นออกจาก CommunicationMessages กรณีไม่มี result จาก getCommunicationMessagesByID จะเรียก getCommunicationMessagesMasterByID แทนแล้วทำการเรียก updateReadState เมื่อонกัน

```
576    @Override
577    Future<Either<Failure, void>> deleteFullRemoteCommunicationMessagesByID(
578        @required String msg_id,
579        @required String myid,
580        String? usecaseName,
581        String? usecaseStep) async {
582        // TODO: implement deleteCommunicationMessagesByID
583
584        try {
585            CoreLog().info("setDeletedMessagesByID ID: $msg_id");
586            var noti = null;
587            try {
588                noti = await localDataSource.getCommunicationMessagesByID(msg_id);
589                // remote delete on sid
590                if (noti != null) {
591                    CoreLog().info(
592                        "Noti DATA madatory: correlationId: ${noti.correlationId} ,TORMD"
593                    var setDeleteNoti = {
594                        "correlationId": [myid], //noti.correlationId,
595                        "messageType": "Mobile app push notification",
596                        "communicationMessage": [
597                            {
598                                "TORMD_messageId": noti.TORMD_messageId,
599                                "TORMD_notificationType":
600                                    noti.TORMD_notificationType //individual
601                            },
602                            "receiver": [
603                                {
604                                    "appUserId": myid,
605                                    } //noti.correlationId
606                                ],
607                                "deleteDate": DateTime.now()
608                                    .toUtc()
609                                    .toString() //"2023-07-05T16:30:04.012Z"
610                            };
611                            var result = await remoteDataSource.updateDeleteState(
612                                setDeleteNoti,
613                                usecaseName: usecaseName,
614                                usecaseStep: usecaseStep,
615                            );
616                            // delete on realm
617                            localDataSource.deleteCommunicationMessagesByID(msg_id);
618                            CoreLog().info("setDeletedMessagesByID Result : $result");
619
620                try {
621                    var noti =
622                        await localDataSource.getCommunicationMessagesMasterByID(msg_id);
623                    if (noti != null) {
624                        var setDeleteNoti = {
625                            "correlationId": [myid], //noti.correlationId,
626                            "messageType": "Mobile app push notification",
627                            "communicationMessage": [
628                                {
629                                    "TORMD_messageId": noti.TORMD_messageId,
630                                    "TORMD_notificationType":
631                                        noti.TORMD_notificationType, //broadcast,
632                                        "TORMD_templateMessage": {
633                                            "id": noti.id,
634                                            "@referredType": "communicationMessageMaster"
635                                        }
636                                },
637                                "receiver": [
638                                    {
639                                        "appUserId": myid,
640                                        } //noti.correlationId
641                                    ],
642                                    "deleteDate": DateTime.now()
643                                        .toUtc()
644                                        .toString() //"2023-07-05T16:30:04.012Z"
645                                };
646                                var result = await remoteDataSource.updateDeleteState(
647                                    setDeleteNoti,
648                                    usecaseName: usecaseName,
649                                    usecaseStep: usecaseStep,
650                                );
651                                CoreLog().info("setDeletedMessagesByID Result : $result");
652                            }
653                        } catch (e) {
654                            ////
655                        }
656                    return const Right(null);
657                }
658            }
659        }
660    }
661
662
663
664
```

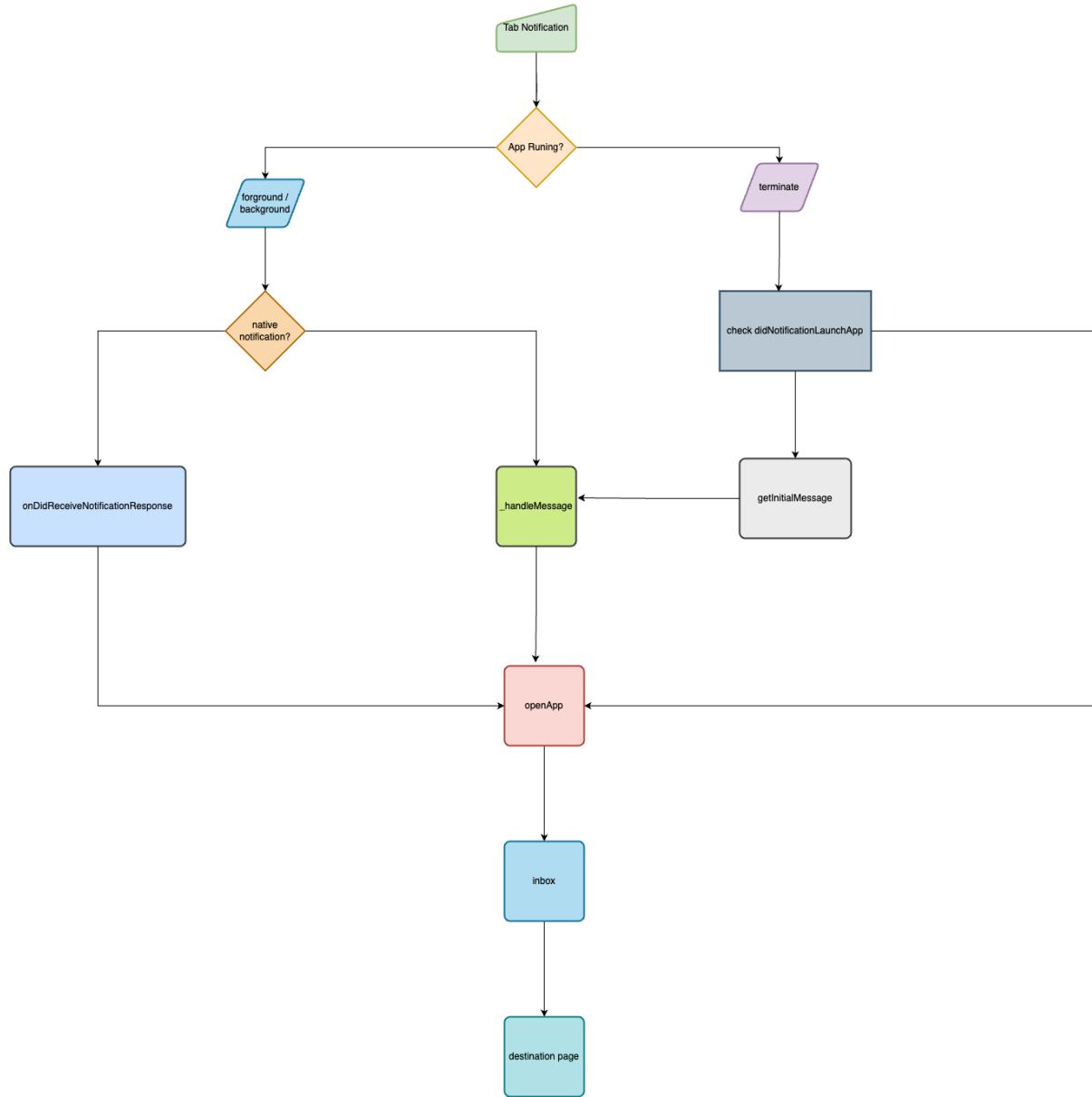
```

86    @override
87    Future updateReadState(Map<String, dynamic> notification,
88        (String? usecaseName, String? usecaseStep)) async {
89        Map<String, String> baseHeaders = {
90            "Accept": "application/json",
91            "Content-type": "application/json; charset=utf-8",
92        };
93        baseHeaders.addAll({
94            HeaderKey.topic.name: "mfaf.updateReadState",
95            HeaderKey.orgService.name: "common.communication",
96            HeaderKey.tmfSpec.name: "TMF681",
97            HeaderKey.baseApiVersion.name: '4.0.0',
98            HeaderKey.eventResponse.name:
99                "mfaf.readStateUpdated,mfaf.readStateUpdateFailed"
100        });
101
102    try {
103        CoreBaseOption coreBaseOption = CoreBaseOption(
104            useCase: usecaseName,
105            useCaseStep: usecaseStep,
106            headers: baseHeaders,
107            body: notification,
108        );
109        CoreLog().info(
110            '++++++updateReadState bodynotification+++++++$({notification})');
111
112        var response = await socketHelper.requestNetwork(coreBaseOption);
113
114        CoreLog().info(
115            '=====updateReadState response datasource===== ${response}');
116        return response;
117    } catch (e) {
118        rethrow;
119    }
120}
121
122    @override
123    Future updateDeleteState(Map<String, dynamic> notification,
124        (String? usecaseName, String? usecaseStep)) async {
125        Map<String, String> baseHeaders = {
126            "Accept": "application/json",
127            "Content-type": "application/json; charset=utf-8",
128            "useCaseName": "",
129            "useCaseStep": ""
130        };
131        baseHeaders.addAll({
132            HeaderKey.topic.name: "mfaf.updateDeleteState",
133            HeaderKey.orgService.name: "common.communication",
134            HeaderKey.tmfSpec.name: "TMF681",
135            HeaderKey.baseApiVersion.name: '4.0.0',
136            HeaderKey.eventResponse.name:
137                "mfaf.deleteStateUpdated,mfaf.deleteStateUpdateFailed",
138        });
139
140    try {
141        CoreBaseOption coreBaseOption = CoreBaseOption(
142            useCase: usecaseName,
143            useCaseStep: usecaseStep,
144            headers: baseHeaders,
145            body: notification,
146        );
147        CoreLog().info(
148            '++++++updateDeleteState bodynotification+++++++$({notification})');
149
150        var response = await socketHelper.requestNetwork(coreBaseOption);
151
152        CoreLog().info(
153            '=====updateDeleteState response datasource===== ${response}');
154        return response;
155    } catch (e) {
156        rethrow;
157    }
158}

```

# Handle Open Notification

การจัดการการเปิด notification เมื่อมีเปิดจากการแตะที่ os notification จะมีการจัดการคร่าวๆตามภาพด้านล่างดังนี้



## App Terminate

เมื่อมีการเปิด notification ขณะที่แอปไม่ได้ทำงาน ตัวเครื่องจะทำการเปิดแอปขึ้นมา หลังจากนั้น initializePushNotificationService จะเริ่มทำเมื่อแอปเข้าสู่หน้า splash

```
681 | Future<void> setUpFCM() async {
682 |   PushNotificationService()
683 |   |   .initializePushNotificationService(pushNotificationContext: context);
684 |   PushNotificationService().onMessage();
685 | }
```

## check didNotificationLaunchApp

หลังจาก initializePushNotificationService เริ่มทำงานจะทำการเช็ค didNotificationLaunchApp ว่ามีการเปิดแอปด้วยการแตะ notification ที่ถูกแสดงด้วย local notification หรือไม่ ถ้าถูกเปิดแอปจาก notification จะทำการดึง notificationResponse ที่เป็น notification payload ออกมาราทำ การ decode เป็น json เพื่อนำ detail ไปใช้งานและเรียกใช้งาน openApp service ต่อไป

```
86 | //check notification detail from localNotificationHelper and decode to send open notification home
87 | if (notificationAppLaunchDetails?.didNotificationLaunchApp ?? false) {
88 |   var details = notificationAppLaunchDetails!.notificationResponse;
89 |   CoreLog().info("Tab local noti open appp, payload = ${details?.payload}");
90 |   var TORO_messageId = "";
91 |   var actionLink;
92 |   if (details?.payload != null) {
93 |     try {
94 |       final NotiDataJson = jsonDecode(details!.payload);
95 |       final contentDetail = jsonEncode(NotiDataJson["contentDetail"]);
96 |       final contentDetailJson = jsonDecode(jsonDecode(contentDetail));
97 |       TORO_messageId = contentDetailJson["TORO_messageId"];
98 |       actionLink = contentDetailJson["actionLink"];
99 |       payload: details.payload,
100 |       firstLaunch: true,
101 |     );
102 |   } catch (e, stack) {
103 |     CoreLog().error(
104 |       "Try to Decode details.payload failed",
105 |       error: e,
106 |       stackTrace: stack,
107 |     );
108 |     PushNotificationService().openApp(
109 |       isForgroud: false,
110 |       firstLaunch: true,
111 |     );
112 |   }
113 | }
114 | }
115 | }
116 | }
117 | }
118 | }
```

## getInitialMessage

หลังจาก initializePushNotificationService เริ่มทำงานจะทำการเช็ค getInitialMessage ว่ามีการเปิดแอปด้วยการแตะ notification ที่ถูกแสดงด้วย native notification หรือไม่ ถ้าถูกเปิดแอปจาก notification จะทำการดึง message ที่เป็น notification payload และเรียกใช้งาน \_handleMessage service ต่อไป

```
120 | //get notification detail from firebase message that open when app is terminate
121 | Future.delayed(const Duration(milliseconds: 100)).then((value) {
122 |   FirebaseMessaging.instance.getInitialMessage().then((message) {
123 |     if (message != null && !haveMessageOpenApp) {
124 |       _handleMessage(message);
125 |     }
126 |   });
127 | });
128 | }
```

## App Running

เมื่อมีการเปิด notification ขณะที่แอพทำงาน ตัวเครื่องจะทำการเปิดแอพขึ้นมาและตัว event listener จะได้รับ event ที่มีการแตะเปิด notification

### Native Notification

เมื่อมีการเปิด notification จะทำให้เกิด event ที่วิ่งมาที่ onMessageOpenedApp ที่มีการเริ่ม listen พร้อมกับตัว initializePushNotificationService ซึ่งจะเรียก \_handleMessage เพื่อทำงานต่อไป

```
68     //firebase message
69     getTokenFromFCM();
70     //onMessage();
71     FirebaseMessaging.onMessageOpenedApp.listen(_handleMessage);
72     FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);
73   
```

### Local Notification

เมื่อมีการเปิด notification จะทำให้เกิด event ที่วิ่งมาที่ LocalNotificationHelper onDidReceiveNotificationResponse ที่มีการเริ่ม listen พร้อมกับตัว initializePushNotificationService ซึ่งจะทำการดึง notificationResponse ที่เป็น notification payload ออกมากำหนด decode เป็น json เพื่อนำ datail ไปใช้งานและเรียกใช้งาน openApp service ต่อไป

```
103     onDidReceiveNotificationResponse: (details) {
104       CoreLog().info(
105         "onDidReceiveNotificationResponse detail : ${details.id}, ${details.actionId},
106
107       var TORO_messageId = "";
108       var actionLink;
109       try {
110         final NotiDataJson = jsonDecode(details.payload!);
111         final contentDetail = jsonEncode(NotiDataJson["contentDetail"]);
112         final contentDetailJson = jsonDecode(jsonDecode(contentDetail));
113         TORO_messageId = contentDetailJson["TORO_messageId"];
114         actionLink = contentDetailJson["actionLink"];
115       } catch (e, stack) {
116         CoreLog().error(
117           "Try to Decode details.payload failed",
118           error: e,
119           stackTrace: stack,
120         );
121       }
122
123       PushNotificationService().openApp(
124         isForgroud: true,
125         TORO_messageId: TORO_messageId,
126         actionLink: actionLink,
127         payload: details.payload,
128       );
129     },
130   );
```

## \_handleMessage

จะทำการแกะ message.data ออกมาเตรียมไว้ แล้วทำการเพิ่ม title, body จาก message.notification  
จากนั้นทำการ decode เพื่อดึงเอา contentDetail, TORO\_messageId,actionLink จากนั้นเรียกใช้  
openApp service ต่อไป

```
198     void _handleMessage(RemoteMessage message) {
199         // do something
200         CoreLog().info(
201             "tab OS noti open app, _handleMessage id = ${message.messageId}");
202
203         try {
204             Map<String, dynamic> notiMap = message.data;
205             notiMap.addAll({
206                 "title": message.notification?.title,
207                 "body": message.notification?.body,
208             });
209
210             final NotiData = jsonEncode(notiMap);
211             final NotiDataJson = jsonDecode(NotiData);
212
213             final contentDetail = jsonEncode(NotiDataJson["contentDetail"]);
214             final contentDetailJson = jsonDecode(jsonDecode(contentDetail));
215
216             final TORO_messageId = contentDetailJson["TORO_messageId"];
217             final actionLink = contentDetailJson["actionLink"];
218
219             if (!haveMessageOpenApp) {
220                 openApp(
221                     isForgrround: false,
222                     TORO_messageId: TORO_messageId,
223                     actionLink: actionLink,
224                     payload: NotiData,
225                     firstLaunch: true,
226                 );
227             } else {
228                 openApp([
229                     isForgrround: true,
230                     TORO_messageId: TORO_messageId,
231                     actionLink: actionLink,
232                     payload: NotiData,
233                     firstLaunch: false,
234                 ]);
235             }
236             haveMessageOpenApp = true;
237         } catch (e) {
238             openApp(isForgrround: false);
239         }
240     }
```

## openApp service

เป็น service สำหรับการเปิดเข้าหน้า inbox โดยจะส่ง parameter ที่จำเป็นเพื่อให้หน้า inbox จัดการ ต่อรวมถึงไปยังหน้าปลายทางตามที่ notification แนบมา

เมื่อเริ่มการทำงานจะทำการเช็คว่าหน้า app ที่อยู่ปัจจุบันเป็นหน้าของ inbox หรือไม่ ผ่าน CoreNavigator().getHistory() และทำการเซ็ต flag [canReplaceRouteNotiHome](#) จากนั้นทำการเช็ค ต่อว่าหน้าปัจจุบันเป็นหน้า main หรือ splash\_screen หรือไม่แล้วทำการเซ็ตflag [openFrom MainPage](#)

```
242     Future<void> openApp(
243         {required bool isForgroud,
244          String? TORO_messageId,
245          String? actionLink,
246          String? payload,
247          bool? firstLaunch}) async {
248
249         //bool canReplaceRouteNotiHome = false;
250         bool openFromMainPage = false;
251         if (TORO_messageId != null && TORO_messageId.trim() != "") [
252             CoreLog().info(
253                 "Try to open notificationHome isForgroud: $isForgroud , TORO_messageId: $TORO_messageId ,actionLink: $actionLink, payload: $payload");
254
255             try {
256                 String? currentPath;
257                 currentPath = CoreNavigator().getHistory().last;
258                 if (currentPath == notificationHome_route_name) {
259                     canReplaceRouteNotiHome = true;
260                 }
261                 if (currentPath == "main" || currentPath == "splash_screen") {
262                     openFromMainPage = true;
263                 }
264             } catch (e) {
265                 CoreLog().error(
266                     "open app notification can not get last of CoreNavigator history with exception: ${e.toString()}",
267                     error: e,
268                 );
269             }
270         }
271     }
```

## Forgroud & HomeAppAlrady

จากนั้นจะทำการเช็คว่ามีการส่ง isForgroud หรือมีการเซ็ตค่า communicationListenerEventService.flagHomeAppAlrady ถ้ามีจะหมายถึงแอพพร้อมที่จะเปิด notification ทันทีจากนั้นจะทำการเช็คflag [canReplaceRouteNotiHome](#) ต่อถ้าหากมีการเซ็ตแล้วจะทำการเปิด หน้า inbox ด้วย [pushReplacementNamed](#) เพื่อไม่ให้หน้า inbox เปิดมากกว่า 1 หน้าซ้อน กัน ถ้าไม่จะ เปิดด้วย [pushNamed](#)

```
271     if (isForgroud || communicationListenerEventService.flagHomeAppAlrady) {
272         if (canReplaceRouteNotiHome) {
273             CoreLog().info("Open notificationHome immediately Replacement");
274             CoreNavigator().pushReplacementNamed(
275                 pushNotificationServiceNavigatorKey.currentState!.context,
276                 notificationHome_route_name,
277                 arguments: {
278                     "targetTOROID": TORO_messageId,
279                     "payload": payload,
280                 }).then((value) {
281                 refresh MainPage(
282                     pushNotificationServiceNavigatorKey.currentState!.context,
283                     openFromMainPage,
284                 );
285             });
286         } else {
287             CoreLog().info("Open notificationHome immediately");
288             CoreNavigator().pushNamed(
289                 pushNotificationServiceNavigatorKey.currentState!.context,
290                 notificationHome_route_name,
291                 arguments: {
292                     "targetTOROID": TORO_messageId,
293                     "payload": payload,
294                 }).then((value) {
295                 refresh MainPage(
296                     pushNotificationServiceNavigatorKey.currentState!.context,
297                     openFromMainPage,
298                 );
299             });
300         }
301     } else {
```

### *Background & Waiting flagHomeAppAlrady*

กรณีที่ isForgroud หรือ communicationListenerEventService.flagHomeAppAlrady ยังไม่มีการ เช็คค่าจะทำการรอ stream จาก communicationListenerEventService.flagHomeAppAlrady จาก นั้นหากมี stream event รึ่งเข้ามาหรือรอเป็นเวลานานกว่า 10 วินาที จะทำการเข้าสู่ขั้นตอนการเปิดหน้า inbox เมื่อ้อนกับการเปิดหน้าของ Forgrround จากนั้นทำการปิด stream ของ flagHomeAppAlrady

```
301     } else {
302         try {
303             await communicationListenerEventService
304                 .alraedyNavigateFlagStream.stream.first
305                 .timeout(
306                     const Duration(seconds: 10),
307                     onTimeout: () {
308                         throw TimeoutException(
309                             "Event from home app did not occur within 10 seconds");
310                     },
311                 );
312             // Event received within 10 seconds
313         } catch (e) {
314             // Timeout occurred
315             CoreLog().info("Event from home app did not occur within 10 seconds");
316         } finally {
317             if (canRepleceRouteNotiHome) {
318                 CoreLog().info("Open notificationHome on stream event Replacemen");
319                 pushNotificationServiceNavigatorKey.currentState!
320                     .pushReplacementNamed("/communication/notificationHome",
321                         arguments: {
322                             "targetToroId": TORO_messageId,
323                             "payload": payload,
324                         }).then((value) {
325                             refresh MainPage(
326                                 pushNotificationServiceNavigatorKey.currentState!.context,
327                                 openFromMainPage,
328                             );
329                         });
330             } else {
331                 CoreLog().info("Open notificationHome on stream event");
332                 pushNotificationServiceNavigatorKey.currentState!
333                     .pushNamed("/communication/notificationHome", arguments: {
334                         "targetToroId": TORO_messageId,
335                         "payload": payload,
336                         }).then((value) {
337                             refresh MainPage(
338                                 pushNotificationServiceNavigatorKey.currentState!.context,
339                                 openFromMainPage,
340                             );
341                         });
342             }
343         }
344         communicationListenerEventService.alraedyNavigateFlagStream.close();
345     }
346 }
347 }
348 }
```

### After open Inbox

ในส่วนของการเปิดหน้า Inbox หรือ notificationHome จะมีการเพิ่มในส่วนของ then เพื่อ handle ต่อ หลังจากที่ออกจาก notificationHome โดยจะเรียก refresh MainPage โดยจะส่ง context และ openFrom MainPage flag เพื่อใช้ในการเช็คว่าจะทำการสั่ง refresh MainPage ของ current tab

```
350     void refreshMainPage(BuildContext context, bool fromMainPage) {
351         if (fromMainPage) {
352             CoreNavigator().tabAnimateTo(context, NavBarName.CURRENT, reload: true);
353         }
354     }
```

### Notification Home

การเปิดหน้า notification home โดยมีการแนบ payload ของ notification มาด้วย จะทำให้มีการทำงานเพิ่มเติมในส่วนของการเปิดอ่านตัว message ให้อัตโนมัตโดย ขั้นตอนการทำงานคร่าวๆ หลังจากเข้ามาที่หน้า notification home และจะทำการแกะ payload และ decode เพื่อใช้งาน ถ้ามีการทำการเช็คว่ามี action link ที่ต้องการเปิดหรือไม่ ถ้ามีจะทำการเช็คต่อว่า target asset ที่แนบมากับ message ตรงกับ current asset หรือไม่ถ้าไม่จะทำการ switch ไปที่ target asset การทำการเปิด action link ด้วย universal open deepLink และทำการ เรียกใช้ usecase setReadedMessagesByToroMessageID เพื่อ set เป็น read

### Initial Notification Service

เพื่อให้ Service ของ notification พร้อมใช้งาน จึงจำเป็นต้องทำการ Initial Notification Service ในส่วนต่างๆ พร้อมกับตัว MFAF App ดังนี้

เมื่อ MFAF App เริ่มทำงานและเริ่ม build widget ของ AppWidget จะทำการเรียก PushNotificationService setPushNotificationServiceNavigatorKey ที่จะรับเอา navigatorKey ที่เป็น GlobalKey<NavigatorState> ของ แอพไปเก็บไว้เพื่อใช้งาน

```
161         return MyPhoneFrame(
162             builder: (context) {
163                 PushNotificationService()
164                     .setPushNotificationServiceNavigatorKey(navigatorKey);
165             }
166         );
167     }
168 }
```

เมื่อ MFAF App เข้าสู่หน้า SplashScreen จะทำการเรียก setUpFCM ที่จะทำการเรียก initializePushNotificationService และ onMessage เพื่อเรียกต้นการทำงาน

```
681     Future<void> setUpFCM() async {
682         PushNotificationService()
683             .initializePushNotificationService(pushNotificationContext: context);
684         PushNotificationService().onMessage();
685     }
686 }
```

## initializePushNotificationService

จะทำการ WidgetsFlutterBinding และ เรียก initializeLangnDevice สำหรับ การตั้งค่า ภาษาของ notification ถัดมาจะทำการ initialCategoryOnDevice เพื่อตั้งค่า setting ของ notification เป็นต้น แล้วทำการ เรียก Firebase.initializeApp() เพื่อเริ่มต้นการทำงานของ FirebaseMessaging ถ้ามีการ ส่ง context จะทำการ set pushNotificationContext เพื่อใช้งาน

```
51 Future initializePushNotificationService(  
52   | {BuildContext? pushNotificationContext}) async {  
53   //Start initializePushNotificationService  
54   WidgetsFlutterBinding.ensureInitialized();  
55   await initialLangnDevice();  
56   await initialCategoryOnDevice("");  
57   await Firebase.initializeApp();  
58  
59   if (pushNotificationContext != null) {  
60     contextPushNotificationService = pushNotificationContext;  
61   }  
62 }
```

## จากนั้นเรียก CommunicationListenerEventService

initializeCommunicationListenerEventService เพื่อเริ่มต้นการทำงานของ app listener

getTokenFromFCM เพื่อดึง token เตรียมให้ core

FirebaseMessaging.onMessageOpenedApp.listen() เปิด listener สำหรับการแตะ notification

FirebaseMessaging.onBackgroundMessage() สำหรับจัดการ BackgroundMessage (ปัจจุบันไม่ได้ใช้งาน )

LocalNotificationHelper.initialize เพื่อเริ่มการทำงานของ LocalNotification เพื่อแสดง notification foreground

ส่วนที่เหลือจะเป็นส่วนของการ Handle Open Notification ในส่วนของ App Terminate

```
63 //start all communicationListenerEventService  
64 //communicationListenerEventService = CommunicationListenerEventService();  
65 communicationListenerEventService  
66   | .initializeCommunicationListenerEventService();  
67  
68 //firebase message  
69 getTokenFromFCM();  
70 //onMessage();  
71 FirebaseMessaging.onMessageOpenedApp.listen(_handleMessage);  
72 FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);  
73  
74 //localNotificationHelper for alert os notification on foreground  
75 LocalNotificationHelper.initialize();  
76 // String notiPermission = await getNotiPermission();  
77 // if (notiPermission == "enable") {  
78 //   FlutterAppBadger.removeBadge();  
79 // }  
80  
81 //get notification detail from localNotificationHelper that open when app is terminate  
82 final NotificationAppLaunchDetails? notificationAppLaunchDetails =  
83   | await LocalNotificationHelper.NOTI_PLUGIN  
84   .getNotificationAppLaunchDetails();  
85 }
```

### *initialCategoryOnDevice*

ทำการ set SharedPreferences ของภาษาจาก CurrentLang().currentLanguage ถ้าหาก error จะใช้ภาษาอังกฤษเป็น default

```
383     Future<void> initialLangnDevice() async {
384         final prefs = await SharedPreferences.getInstance();
385         try {
386             prefs.setString(
387                 prefs_key_notification_lang, CurrentLang().currentLanguage);
388             CoreLog().info(
389                 "initialLangnDevice : ${prefs.getString(prefs_key_notification_lang)} ");
390         } catch (exception) {
391             prefs.setString(prefs_key_notification_lang, "en");
392         }
393     }
```

### *initialCategoryOnDevice*

ทำการตั้ง prefs\_key\_installAppDatetime จาก SharedPreferences ว่ามีการบันทึกค่าวันที่ติดตั้งแอพหรือยัง ถ้ายัง บันทึกวันที่ปัจจุบันลงไป และ set notification setting ด้วย

**default\_category\_push\_notifiaction**

กรณีที่มีการบันทึกค่าวันที่ติดตั้งแอพแล้วจะทำการตั้งค่า notification setting เพื่อขึ้นมาแล้วบังคับเปิด bill category และบันทึกค่าลงไปใหม่

```
395     Future<void> initialCategoryOnDevice(String fcmtoken) async {
396         // AllowedNotificationCategoryLocalDataSourceImpl allocatesLocalDataSource =
397         //     AllowedNotificationCategoryLocalDataSourceImpl();
398         final prefs = await SharedPreferences.getInstance();
399         try {
400             //check first time install app via prefs_key_installAppDatetime
401             if (prefs.getString(prefs_key_installAppDatetime) == null) {
402                 CoreLog()
403                     .info("first time install app. begin initialCategoryOnDevice ");
404                 await prefs.setString(
405                     prefs_key_installAppDatetime,
406                     DateFormat('yyyy-MM-dd@HH:mm:ss:ms')
407                         .format(DateTime.now().toUtc()));
408
409                 await prefs.setStringList(prefs_key_allowedNotificationCategory,
410                     default_category_push_notifiaction);
411             } else {
412                 var perfAllowedCategoryNotification =
413                     prefs.getStringList(prefs_key_allowedNotificationCategory) ??
414                     default_category_push_notifiaction;
415
416                 //force add bill category
417                 if (!perfAllowedCategoryNotification
418                     .contains(bill_category_push_notifiaction)) {
419                     perfAllowedCategoryNotification.add(bill_category_push_notifiaction);
420                     await prefs.setStringList(prefs_key_allowedNotificationCategory,
421                         perfAllowedCategoryNotification);
422                 }
423             }
424         } catch (e, stack) {
425             CoreLog().error(
426                 "Exception on initialCategoryOnDevice",
427                 error: e,
428                 stackTrace: stack,
429             );
430         }
431     }
```

## LocalNotificationHelper

เป็นการเริ่มต้นการทำงานของ service จัดการ notification foreground

```
59  class LocalNotificationHelper {
60    static final FlutterLocalNotificationsPlugin NOTI_PLUGIN =
61      FlutterLocalNotificationsPlugin();
62
63    static const AndroidNotificationChannel androidNotificationChannel =
64      AndroidNotificationChannel(
65        "mfaf_push_noti",
66        "MFAF_PUSH_NOTI",
67        description: "mfaf channel for androidNotification",
68        importance: Importance.max,
69      );

```

## initialize

สร้าง android setting โดยจะเริ่มจากกำหนด icon app ของ notification ที่จะใช้แสดงของ android ผ่าน AndroidInitializationSettings("@mipmap/ic\_launcher")

DarwinInitializationSettings เป็น ios setting ซึ่งจะใช้ตาม default จาก lib จากนั้นทำการ InitializationSettings ของ lib ด้วย setting ของ android ios ที่เตรียมไว้

```
71  static void initialize() {
72    const AndroidInitializationSettings androidInitializationSettings =
73      AndroidInitializationSettings("@mipmap/ic_launcher");
74
75    final DarwinInitializationSettings initializationSettingsDarwin =
76      DarwinInitializationSettings(
77        requestAlertPermission: false,
78        requestBadgePermission: false,
79        requestSoundPermission: false,
80        onDidReceiveLocalNotification:
81          (int id, String? title, String? body, String? payload) async {
82            didReceiveLocalNotificationStream.add(
83              ReceivedNotification(
84                id: id,
85                title: title,
86                body: body,
87                payload: payload,
88                ), // ReceivedNotification
89              );
90            },
91          ); // DarwinInitializationSettings
92
93    final InitializationSettings initializationSettings =
94      InitializationSettings(
95        android: androidInitializationSettings,
96        iOS: initializationSettingsDarwin,
97      );

```

ทำการเรียก createChannel สำหรับสร้างช่องทางส่งไปยังเครื่องของ android จากนั้นสร้าง initialize ของตัว FlutterLocalNotificationsPlugin เพื่อใช้งานต่อไป

```
99  createChannel(androidNotificationChannel);
100
101 NOTI_PLUGIN.initialize(
102   initializationSettings,
103   onDidReceiveNotificationResponse: (details) {
104     CoreLog().info(
105       "onDidReceiveNotificationResponse detail : ${details.id}, ${details.actionId},
106
107       var TORO_messageId = "";
108       var actionLink;
109       try {
110         final NotiDataJson = jsonDecode(details.payload!);
111         final contentDetail = jsonEncode(NotiDataJson["contentDetail"]);
112         final contentDetailJson = jsonDecode(jsonDecode(contentDetail));
113         TORO_messageId = contentDetailJson["TORO_messageId"];
114         actionLink = contentDetailJson["actionLink"];
115       } catch (e, stack) {
116         CoreLog().error(
117           "Try to Decode details.payload failed",
118           error: e,
119           stackTrace: stack,
120         );
121       }
122
123       PushNotificationService().openApp(
124         isForeground: true,
125         TORO_messageId,
126         actionLink: actionLink,
127         payload: details.payload,
128       );
129     },
130   );
131 }
```