COMP7510
Internet Computing and Programming

Lab Manual 2 – Accessing Firebase Database and Storage

Part 4 – Firebase Service

Firebase is a service provided by Google. It provides us with functionalities including database, file storage, analytics, messaging, and so on. In our labs, we only focus on the database and file storage. With **Firebase**, our data can be stored online and accessed anywhere without maintaining the backend system with data storages. No backend system is maintained by ourselves, so we need to pay more attention to the data saving part of our program codes. Once the data are deleted or overwritten, they cannot be recovered.

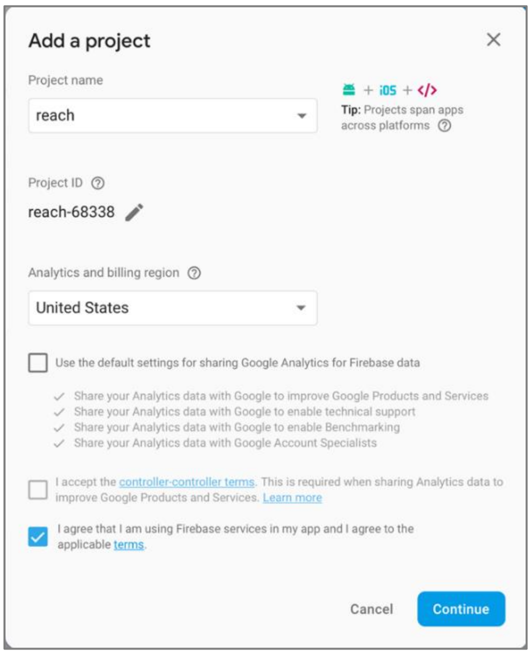
Enabling Firebase Service

To enable the **Firebase** service, we need a Google account. We can directly use our BU campus email account because it is a Google account.

Let's follow the steps below to enable **Firebase** service:

1. Go to the URL – https://console.firebase.google.com. If you have not yet logged in, log in with your BU campus email account or your personal Google account.

2. Click "Add Project".



Add a project

Project name: reach

Tip: Projects span apps across platforms

Project ID: reach-68338

Analytics and billing region: United States

[ ] Use the default settings for sharing Google Analytics for Firebase data
- ✓ Share your Analytics data with Google to improve Google Products and Services
- ✓ Share your Analytics data with Google to enable technical support
- ✓ Share your Analytics data with Google to enable Benchmarking
- ✓ Share your Analytics data with Google Account Specialists

[ ] I accept the controller-controller terms. This is required when sharing Analytics data to improve Google Products and Services. Learn more

[✓] I agree that I am using Firebase services in my app and I agree to the applicable terms.

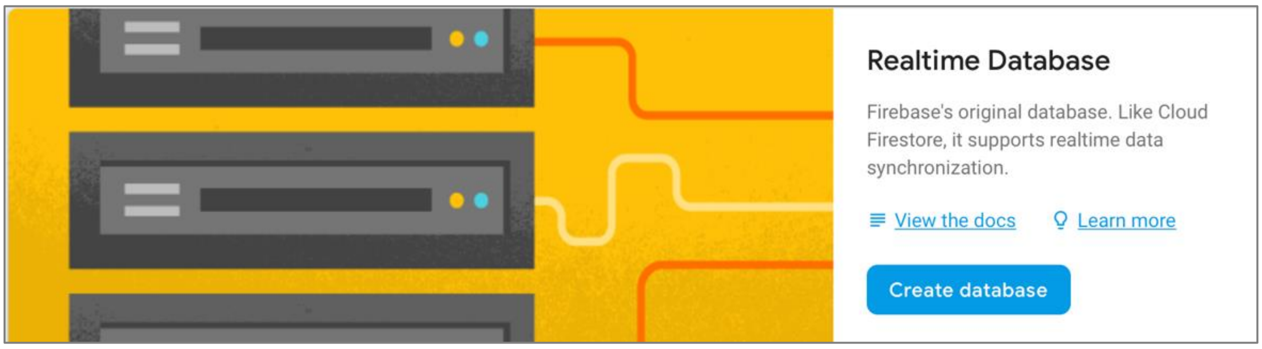
Cancel | Continue

3. On the pop-up panel, type “reach” for the project name, uncheck the option of Google Analytics and check the option of the agreement. Then, click “Continue”.

4. Click “Create Project” shown in the bottom right corner of the second page of the pop-up panel.

5. When a message “Your new project is ready” shown, click “Continue”. You then should receive a Welcome email from **Firebase**.

6. Go back to the **Firebase** console. In the left panel, expand the “Develop” list and select “Database”.



Realtime Database

Firebase's original database. Like Cloud Firestore, it supports realtime data synchronization.

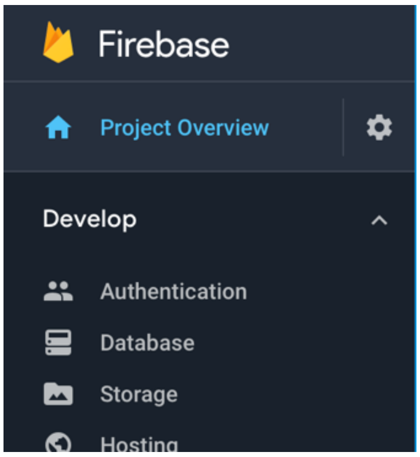
7. In the main panel, scroll down to find **Real-time Database**, and then click “Create database”.

8. On the pop-up panel, select “Start in test mode” and click “Enable”.

9. In the left panel, click “Storage”. Then, in the main panel, click “Get Started”.

10. On the pop-up panel, click “Got it”.

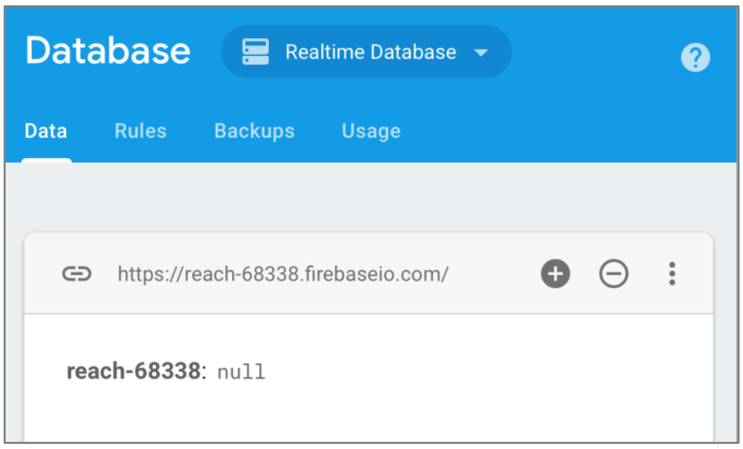
11. Now, the database and storage features are ready.



Data in Firebase Database

After creating the database, we can manage our data through the **Firebase** console. To view your database, you can:

- Click the "Database" in the left panel. Then, add "/data" to the end of the URL.



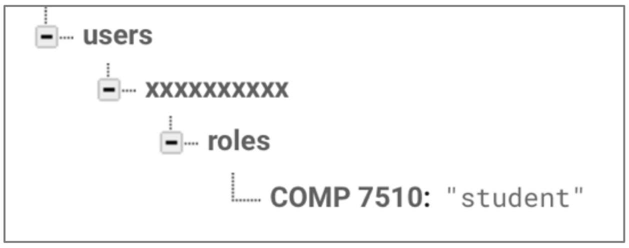
```
reach-68338: null
```

Manual Input

Next, we are going to add the data to the database manually. Let's follow the steps below:

1. Move the mouse pointer over the root entry (**reach-XXXX** entry). Then, click its "+" sign to add a child entry.
2. Type "users" for the **name** field, and then click its "+" sign to add a child entry.
3. Type your student ID for the **name** field, and then click its "+" sign to add a child entry.
4. Type "roles" for the **name** field, and then click its "+" sign to add a child entry.
5. Type "COMP 7510" for the **name** field, and type "student" for the **value** field.
6. Then, click "Add" to commit the creation.

✪ *Click the "X" sign if you want to delete an entry. All its child entries will be deleted too. And, no undo service is provided.*



```
users
  xxxxxxxxxx
    roles
      COMP 7510: "student"
```

Data Import

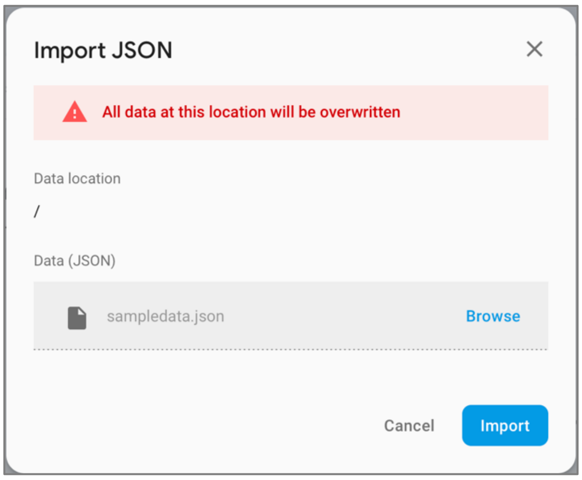
If you have a JSON data file on hand, you may import the data from the data file to your database. Let's follow the steps below to import the data:

1. Download the sample data file – **sampledata.json**, from our course web page.

2. Click the rotated "…" button (more button) at the top right corner of the main panel, and select "Import JSON".

3. Click "Browse" and select the **sampledata.json** file. Then, click "Import" to commit.

4. Now, you have the data imported.



Data Export

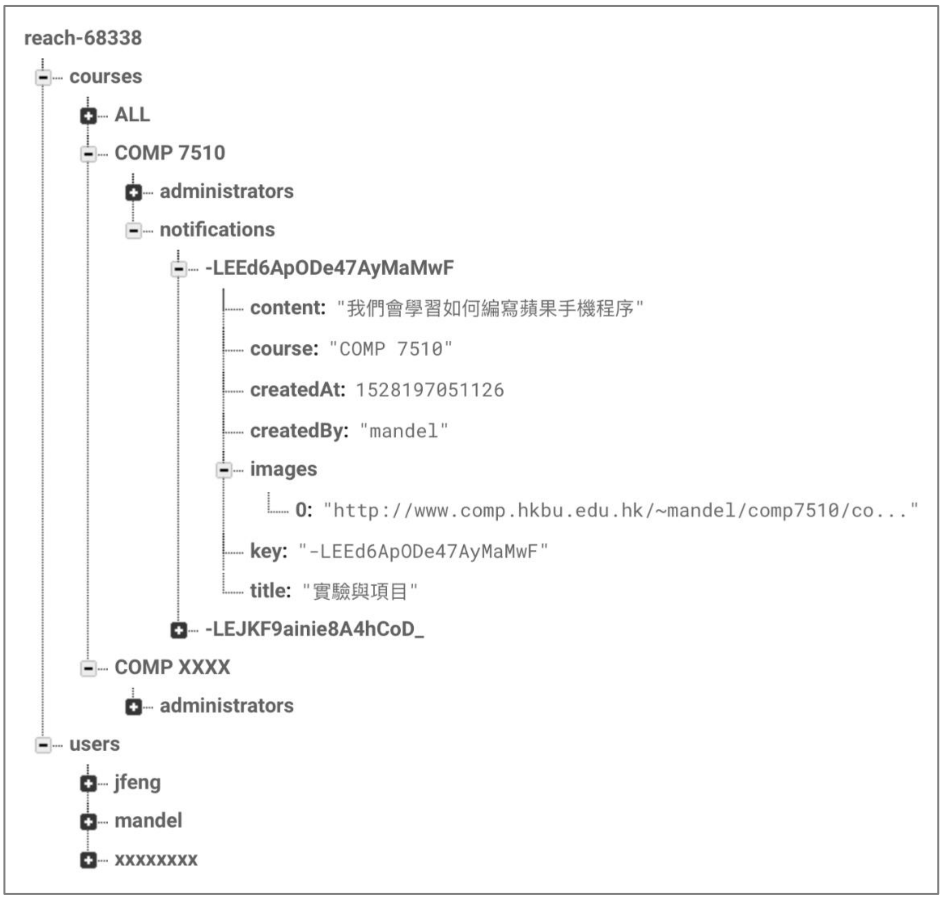
If you want to make a backup copy of your existing data, you may use the export function by selecting "Export JSON" in the menu of the more button.

Data Presentation

Now, the data are ready but we need to understand how they are presented in the **Firebase** database. Some points you need to pay attention:

- **Firebase** database is a NoSQL database. Data are not separated in different tables as a relational database.
- The data in **Firebase** database are shown as a tree structure.
- A data entry is represented by key and value. The key is used to refer to the entry.
- A key must be in string format. A value can be a string, number, or map structure.
- Values may be duplicated. Denormalization is often in **Firebase** database.
- There is no control for adding sub-entries to an entry.
- **Firebase** database supports Unicode, so you can store Chinese words and symbols.
- The long text will not be displayed completely in the console, but there is no problem with your app.

Consider the following sample data segment:



```
reach-68338
  courses
    ALL
    COMP 7510
      administrators
      notifications
        -LEEd6ApODe47AyMaMwF
          content: "我們會學習如何編寫蘋果手機程序"
          course: "COMP 7510"
          createdAt: 1528197051126
          createdBy: "mandel"
          images
            0: "http://www.comp.hkbu.edu.hk/~mandel/comp7510/co..."
          key: "-LEEd6ApODe47AyMaMwF"
          title: "實驗與項目"
        -LEJKF9ainie8A4hCoD_
    COMP XXXX
      administrators
  users
    jfeng
    mandel
    xxxxxxxx
```

- The **reach-68338** entry is the root of the database. It has two child entries – **courses** and **users**.

- The **courses** entry has three child entries including **ALL**, **COMP 7510** and **COMP XXXX**.

- The **notifications** entry of the COMP 7510 entry contains two child entries **-LEEd6ApODe47AyMaMwF** and **-LEJKF9ainie8A4hCoD_.** These notifications are about the course COMP 7510. We express that the notifications will be read by COMP 7510 teachers and students only.

- **-LEEd6ApODe47AyMaMwF** is a key referring to a notification. The notification contains the fields including title, content, course, created time, author, image links, a copy of the key. The keys – "title", "content", "course", "createdAt", "createdBy", "images", and "key" refer to these fields respectively.

- The **key** entry is a copy of the key of the **notification** entry that will be used for making our program shorter and run faster.

- The notification **-LEEd6ApODe47AyMaMwF** has an **images** entry; The another one **-LEJKF9ainie8A4hCoD_** does not have an **images** entry. This is allowed in **Firebase** database, but you need to handle this issue in your app.

- The **COMP XXXX** entry does not have the **notifications** entry, which indicates no notification has been posted for COMP XXXX yet.

Rules

When we create the database, we selected the option – "Start in test mode". Now everyone can read and write our databases through the database URL.

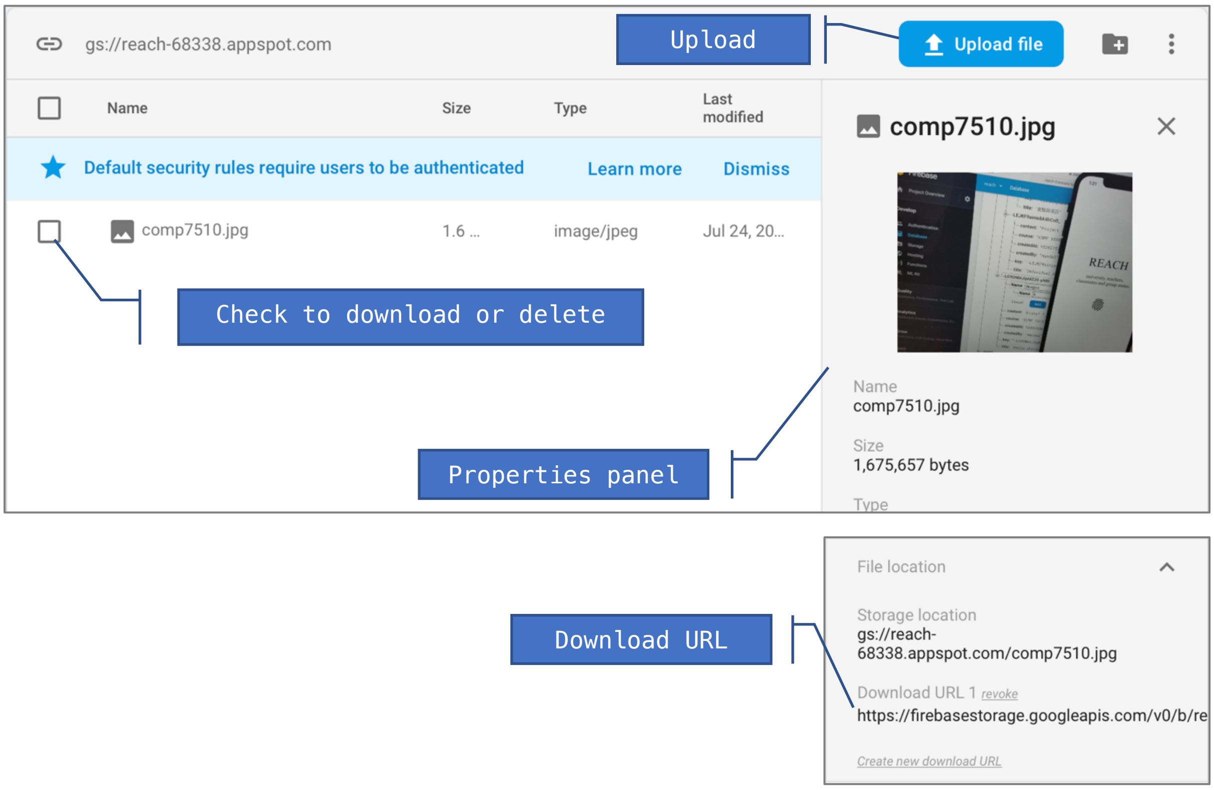
Now, we set the minimum security to the database by publishing the new rules as follows:

```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

Of course, we should not dispose our database URL. Or, use an advanced authentication method. But, the advanced authentication is out of our scope.

Files in Firebase Storage

Firebase Storage is a simple file online storage that allows us to upload, download and delete files. To upload a file, we can simply click on the "Upload file" button at the top right corner of the "Storage" panel. After uploading a file, you can check its detailed information by clicking on the file item. In its properties panel, you can find a download URL for downloading the file. Or, you can check the file items to download files in batch. You can also delete the selected files.



gs://reach-68338.appspot.com

Name	Size	Type	Last modified
comp7510.jpg	1.6 ...	image/jpeg	Jul 24, 20...

Default security rules require users to be authenticated

comp7510.jpg

Name
comp7510.jpg

Size
1,675,657 bytes

Type

File location

Storage location
gs://reach-68338.appspot.com/comp7510.jpg

Download URL 1
https://firebasestorage.googleapis.com/v0/b/re

Google Sign-in & Firebase Authentication

As we enabled the authentication requirement of the database and the default security of the storage requires user authentication, our app requires an authentication procedure for **Firebase** service. Otherwise, our app cannot read and write the database and storage. **Firebase** accepts different authentication methods including username/password login, Google sign-in, and Facebook sign-in. To limit the usage of our app for HKBU's students and staffs only with avoiding additional username and password, we will use Google sign-in (remember that our campus email accounts are Google accounts).

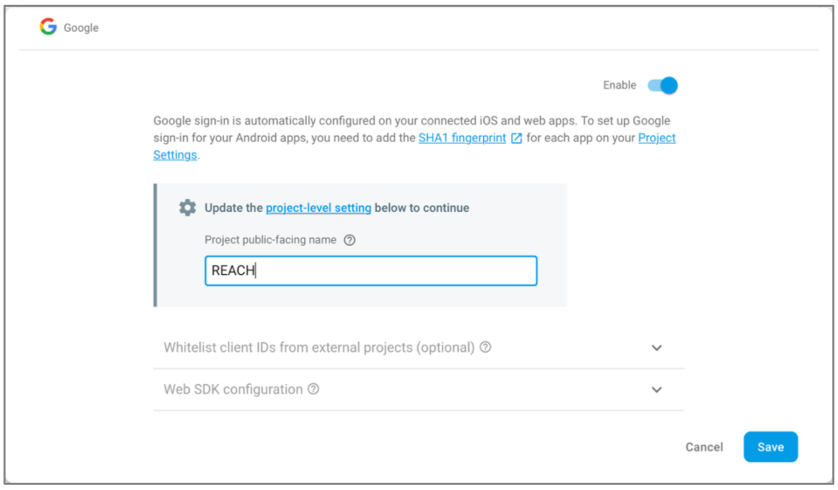
Let's follow the steps below to enable the Google authentication for **Firebase**:

1. Click "Authentication" in the left panel.

2. In the "Authentication" panel, click "Sign-in method", and select "Google" sign-in provider.

3. Click the "Enable" button to enable the Google sign-in provider. And, try "REACH" for *Public-facing name*.

4. Click "Save" to commit.



Google sign-in is automatically configured on your connected iOS and web apps. To set up Google sign-in for your Android apps, you need to add the SHA1 fingerprint for each app on your Project Settings.

Update the project-level setting below to continue

Project public-facing name

REACH

Part 5 – Adding Firebase to iOS App

The next procedure is to let our app access **Firebase**. Our app requires **Firebase** SDK so that it can access the **Firebase** database and storage.

Installing Firebase SDK

Let's follow the steps below:

1. Use Finder to locate the **Runner.xcworkspace** file in the **Documents/reach/ios/** folder.

2. Double-click to open the **Runner.xcworkspace** file. **Xcode** then will be launched.

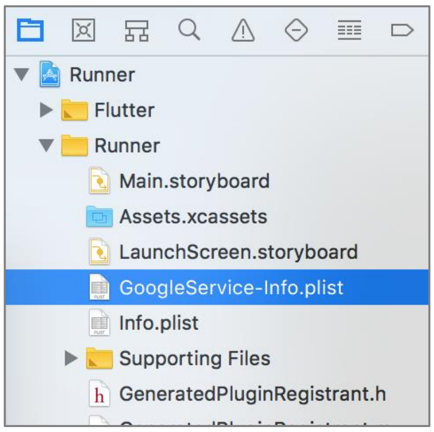
3. Use a web browser to open **Firebase** console, click "Project Overview".

4. Click "Add **Firebase** to your iOS app".

5. In the "Add **Firebase** to your iOS app" page, try **hk.edu.hkbu.comp.reach** for the iOS bundle ID. Then, click "Register App".

6. Click "Download GoogleService-Info.plist" to download the **GoogleService-Info.plist** file.

7. Drag and drop the **GoogleService-Info.plist** file to the *Project Navigator* of **Xcode** under the **Runner** sub-folder.



8. In the pop-up window, enable the option "Destination: copy items if needed" and click "Finish".

9. Close **Xcode**. And skip the remaining steps of the "Add **Firebase** to your iOS app" page.

10. Go back to **IntellliJ**, open the **Info.plist** file located in the **/reach/ios/Runner** folder. Add the following lines before the *</dict>* tag.

```
<key>CFBundleURLTypes</key>
   <array>
       <dict>
          <key>CFBundleTypeRole</key>
          <string>Editor</string>
          <key>CFBundleURLSchemes</key>
          <array>
             <string>xxxxxxxxxxxx</string>
          </array>
       </dict>
   </array>
```

✪ *xxxxxxxxxxxx should be replaced with your own reversed client ID. You can find it in your **GoogleService-Info.plist** file.*

11. Open the **pubspec.yaml** file & add the following lines to the "dependencies" section.

```
google_sign_in: ^3.0.4              # for google sign-in
firebase_auth: ^0.5.15              # for Firebase sign-in
firebase_database: ^0.4.6           # for database access
firebase_storage: ^0.3.7            # for storage access
```

12. Save the file and click the "Package get" link.

Firebase Initialization

The project setting is now ready. Next, we need to add program codes. We need to declare some variables and initiate the **Firebase** module. Let's add the following codes to the **global.dart** file.

1. Import the necessary packages.

```
import 'package:google_sign_in/google_sign_in.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_database/firebase_database.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:fluttertoast/fluttertoast.dart';
```

2. Declare some global variables and a function named **firebaseInit()** for initiating the **Firebase** module. Add the following codes to the end of the **global.dart** file.

```
GoogleSignIn googleSignIn;
FirebaseAuth firebaseAuth;
DatabaseReference dbRef;
StorageReference storageRef;
String userID;
Map roles;

void firebaseInit() {
  var firebaseApp = FirebaseApp.instance;
  googleSignIn = GoogleSignIn(scopes: ['email']);
  firebaseAuth = FirebaseAuth.instance;

  dbRef = FirebaseDatabase(app: firebaseApp).reference();
  storageRef =
    FirebaseStorage(app:firebaseApp, storageBucket:'yyyyyyyyyy').ref();

  Fluttertoast.showToast(msg: 'Initialization is done');
}
```

✪ ***yyyyyyyyyy*** *should be replaced with the URL of your* ***Firebase*** *storage. You can find it in* ***Firebase*** *console.*

3. Open the **main.dart** file and change the **main()** function as follows:

```
void main() {
  firebaseInit();
  runApp(MyApp());
}
```

Sign-in

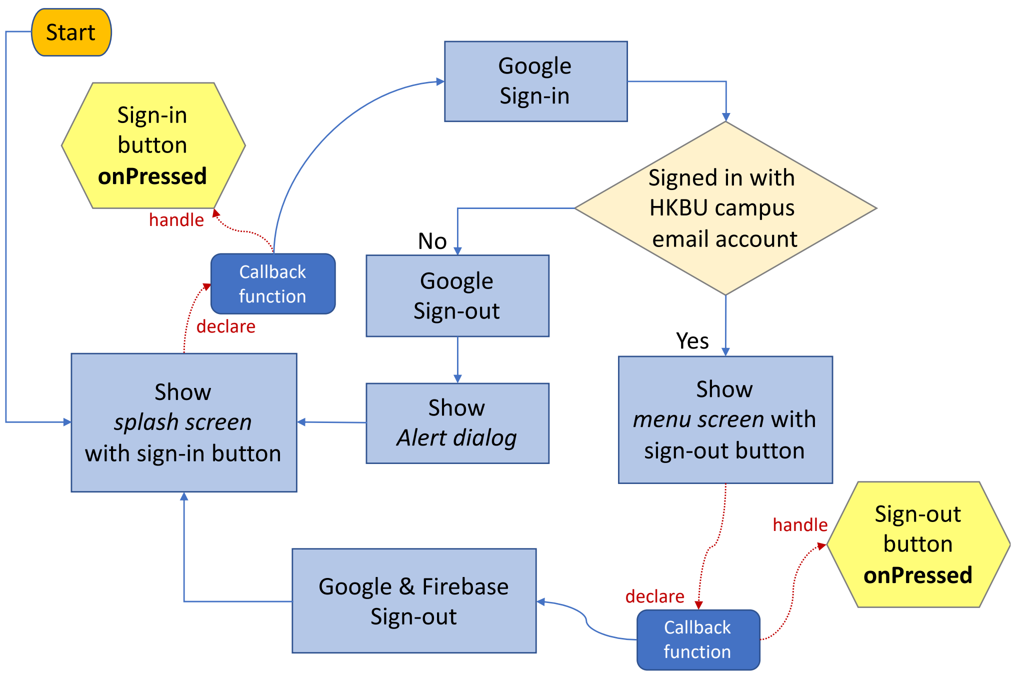
Next, we add the **signIn()** function for performing the sign-in procedure. The **signIn()** function will be invoked when the user presses the icon button of the Home page. The sign-in logic is as follows:

i. Show the splash screen with the sign-in button. The **On-Pressed** event is triggered if the user provides the sign-in button.

ii. Then, the **GoogleSignIn.signIn()** method will be invoked, and the Google sign-in page prompts.

iii. If the user signs in with HKBU campus email account, the account information (from Google sign-in API) will be used for **Firebase** authentication. The user ID will be stored for further usage in the app. The main menu screen shows as well.

iv. If the user does not sign in with HKBU campus email account, an alert page shows about the problem.


Start
Sign-in button onPressed
handle
Callback function
declare
Google Sign-in
Signed in with HKBU campus email account
No
Google Sign-out
Yes
Show splash screen with sign-in button
Show Alert dialog
Show menu screen with sign-out button
handle
Sign-out button onPressed
declare
Google & Firebase Sign-out
Callback function

Let’s add the following procedures to add the sign-in logic:

1. Add the codes in the **global.dart** file:

```
1   Future<bool> signIn(context) async {
2
3     var account = await googleSignIn.signIn();
4
5     if (account != null && account.email.endsWith('hkbu.edu.hk')) {
6
7       var googleAuth = await account.authentication;
8       var user = await firebaseAuth.signInWithGoogle(
9           idToken: googleAuth.idToken,
10          accessToken: googleAuth.accessToken
11      );
12      userID = user.email.substring(0, user.email.indexOf('@'));
13
14    } else {
15
16      googleSignIn.signOut();
17      userID = null;
18
19      var alert = AlertDialog(
20        title: Text('Sign in'),
21        content: Text('Please sign in with your HKBU email account.'),
22      );
23
24      showDialog(context: context, builder: (_)=>alert);
25    }
26
27    return userID != null;
28  }
```

- Declare an asynchronized method, **signIn()** that performs Google sign-in by using the **googleSignIn.signIn()** method (line 3). The **await** expression forces the program waiting for the completion of the method.

- Check whether the user signed in with HKBU email account (line 5). If yes, sign in **Firebase** using the account information (line 7 ~ 12).
- Otherwise, perform Google sign-out (line 16). And, show an alert dialog about the sign-in requirement (line 19 ~ 24).

- Finally, return the sign-in result (true = signed in; false = not signed in).

2. Open the **home.dart** file. In the **splashScreen()** method, change the callback function of the **IconButton** widget as follows:

```
      Widget splashScreen() {
        return Scaffold(
            appBar: null,
            body: Container(
              width: MediaQuery.of(context).size.width,
              child: Column(
                  …

                  IconButton(
                    icon: Icon(Icons.fingerprint),
                    iconSize: 64.0,
➔                     onPressed: () => signIn(context).then((success){
➔                       if (success) setState((){});
➔                     }),
                      …
      }
```

- Invoke the **signIn()** method. Then, we call the **setState()** method with empty body if the result of the **signIn()** is true.

3. As we declare variable **userID** in the **global.dart** file. It can be accessed anywhere. So, we need to delete the one declared inside the **HomeState** class. And, add the import statement at the top of the **home.dart** file to import the **global.dart** file.

```
➔   import 'global.dart';
    …

    class HomeState extends State {
➔     // var userID = 'hello';

      @override
      Widget build(BuildContext context) { … }
```

4. Save the files and re-run the app.

5. In the splash screen, press the fingerprint icon to sign in. The Google sign-in screen will be shown.

6. Type your HKBU email address and password to sign in. Then, you should see the main menu screen.

Sign-out

Comparing with the sign-in procedure, the sign-out procedure is much simpler. We only need to set **userID** to null and call the **signOut()** methods of **firebaseAuth** and **googleSignIn**.

1. Add the following codes to the end of the **global.dart** file:

```
void signOut() async {
  userID = null;
  await firebaseAuth.signOut();
  await googleSignIn.signOut();
}
```

2. Open the **home.dart** file, go to the **menuScreen()** method. And, change the callback function of the icon button nested in the app bar as follows:

```
      Widget menuScreen() {
        return Scaffold(
          appBar: AppBar(
            title: Text('REACH'),
            actions: <Widget>[
              IconButton(
                icon: Icon(Icons.account_box),
➔               onPressed: (){
➔                 signOut();
➔                 setState((){});
➔               },
              ),
            ],
          ),
          …
```

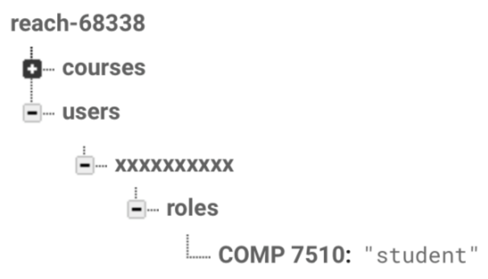
3. Save the files and re-run your app. Then, click the icon button on the main menu screen to sign out.

Retrieving Data from Firebase Database

The **DatabaseReference** class is used for representing a particular position in the **Firebase** database. In the initialization of the **Firebase** module, we set variable **dbRef** to the root of our database using the following statement.

```
dbRef = FirebaseDatabase(app: firebaseApp).reference();
```

Its **child()** method is used to specify another position under the current position. Consider the following sample data segments stored in the **Firebase** database:



```
reach-68338
├── courses
└── users
      └── xxxxxxxxxx
            └── roles
                  └── COMP 7510: "student"
```

- **dbRef** represents the root of the database – *reach-68338*.

- To represent my roles, we do:
```
DatabaseReference rolesRef =
dbRef.child('users/mandel/roles');
```

After we set a data reference to a position, we can get the data by the data reference. We discuss two **Firebase** database API methods – blocking method and listener method.

Blocking method – once()

The **once()** method is an asynchronized method that returns a future snapshot object. The method call is a one-time operation. For example, we need to retrieve the roles of a user. So, we do:

```
1  Future<void> getRoles() async {
2
3    var rolesRef = dbRef.child('users/$userID/roles');
4    var snapshot = await rolesRef.once();
5    roles = snapshot.value as Map;
6
7  }
```

- The data reference **rolesRef** points to the position '*users/$userID/roles*' (line 3). Assume that variable **userID** is storing an ID of a user, and *$userID* in the string will be replaced with the value of variable **userID**.

- The **once()** method is an asynchronized method to download the data from the database (line 4).

    - The **await** expression is added to wait until the asynchronized data download operation completes.

    - We have to use an asynchronized method (a method declared with **async** expression) to store the statements if we use the **await** expression.

    - The **once()** method returns a **DataSnapshot** object.

- We retrieve the value of the **DataSnapshot** object, convert it to a Map, and assign it to the global variable **roles** (line 5).

    - If **userID** equals 'mandel', variable **roles** will be:
      { 'ALL' : 'administrator', 'COMP 7510' : 'teacher', 'COMP XXXX' : 'administrator' }

If we do not use the **await** expression, we can use **then()** method with a callback function to retrieve the data.

```
void getRoles() {

  var rolesRef = dbRef.child('users/$userID/roles');

  rolesRef.once().then((snapshot) =>  roles = snapshot.value as Map);

}
```

Listener method – onValue.listen()

The **onValue** property of the data reference points to a stream connecting to the database. We retrieve the data by attaching a listener to the stream. The listener is triggered once for the initial state of the data and again anytime the data changes.

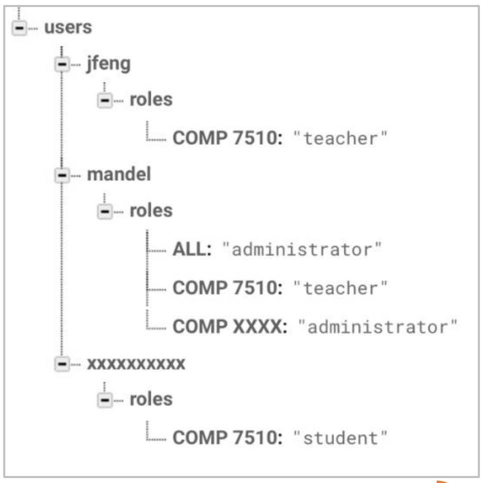
The following is the example codes to retrieve roles using listener method:

```
void getRoles() {

  var rolesRef = dbRef.child('users/$userID/roles');

  rolesRef.onValue.listen((event){
    roles = event.snapshot.value as Map;
  });

}
```

Retrieving Notifications in Our App

Imagine that we have many courses and each course has its notifications for its teachers and students. Now, a student takes COMP7510 only, he/she should be able to read the notifications of COMP7510 and the public notifications (for all people). Of course, he/she should not be able to read the notifications of other courses. The same idea is also applicable to the teachers and administrators who work for different courses.

The child entries of the roles entry store the user roles of different courses, and the course codes are used as keys of the entries. These keys can be used to determine which course notifications should be shown to the users.



```
users
  jfeng
    roles
      COMP 7510: "teacher"
  mandel
    roles
      ALL: "administrator"
      COMP 7510: "teacher"
      COMP XXXX: "administrator"
  xxxxxxxxxx
    roles
      COMP 7510: "student"
```

Maps

A map is a structure that manages the values by using keys. Both keys and values can be any type of objects. Each key occurs only once, but the map can store the same value multiple times.

A map can be created and initialized with a set of elements with keys contained in a set of braces "{ }", or using the constructor of the **Map** class – **Map<K, V>()**.

Declaration:

```
var map1 = {
   'Alice' : '852-66558899',

   'Bob' : '852-98745612',

};

var map2 = Map<String, String>();
```

Adding / setting an element:

```
map1['Cathy'] = '852-56789012';
```

Deleting an element:

```
map1.remove('Bob');
```

Access an element:

```
var phone = map1['Alice'];
```

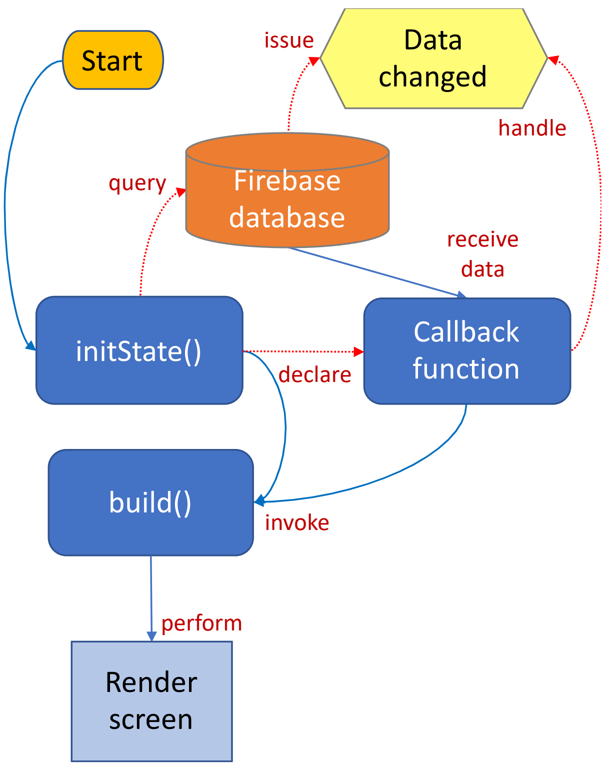
Adding Code

The flow of retrieving the roles and notifications is as follows:

1. Before constructing the screen, during the initial state, we send queries to the **Firebase** database for retrieving the user roles and notifications.

2. The **build()** method is invoked then to construct the screen with an empty list view because the data is not yet delivered.

3. The callback function will be invoked when the data is ready. The callback function calls the **build()** method again to reconstruct the screen with the newly received data.



Let's follow the steps below to add the codes to the app:

1. Add the following method to the **global.dart** file:

```dart
Future<void> getRoles() async {
  var rolesRef = dbRef.child('users/$userID/roles');
  var snapshot = await rolesRef.once();
  roles = snapshot.value as Map;
}
```

2. Open the **notification_list.dart** file and put the following line to the top of the file.

```dart
import 'package:intl/intl.dart';
```

3. put the following codes to the top of the **NotificationListState** class:

```
1   var canCreate = false;
2   var nMap = {};
3
4   void getNotificationList() {
5     Set roleSet, courseSet;
6     if (roles != null) {
7       roleSet = roles.values.toSet();
8       courseSet = roles.keys.toSet();
9     } else {
10      roleSet = Set();
11      courseSet = Set();
12    }
13    courseSet.add('ALL');
14    canCreate = roleSet.contains('teacher')
15      || roleSet.contains('administrator');
16
17    for (var c in courseSet) {
18      var nRef = dbRef.child('courses/$c/notifications');
19
20      nRef.onValue.listen((event) {
21        if (event.snapshot.value == null) nMap.remove(c);
22        else nMap[c] = (event.snapshot.value as Map).values.toList();
23
24        if (mounted) setState(() {});
25      });
26    }
27  }
```

- Variable **canCreate** represents whether the user can create new notifications or not (line 1). Variable **nMap** is a map for storing the notifications downloaded from the **Firebase** database (line 2).

- The **getNotificationList()** method is used to retrieve the notifications from different courses.

- The global variable "**roles**" stores the user roles – course codes are keys, and roles are values. We separate the keys and the values and store them as *Sets* (line 5 ~ 12). If the global variable "**roles**" equals to *null*, it indicates that there is no information about the user roles in the Firebase database.

- A new element 'ALL' is added to the course set. It indicates the user can read the public notifications. (line 13).

- The **contains()** method is used to check whether **roleSet** contains 'teacher' or 'administrator'. The Boolean result is stored in variable **canCreate** that will be used later (line 14 ~ 15).

- The *for-in* loop reads each element of the course set and creates a listener to the data references for listening to the data changes about the notifications (starting from line 17).

- Listeners and callback functions are declared for listening and handling the data changes of different courses. Once the data change occurs, the callback function (line 20 ~ 25) will be invoked. In the callback function we do the following actions:

- If the snapshot contains nothing (no notification for the course), we delete the old notifications about the course. Otherwise, we use the new set of notifications to replace the existing set stored in the map.

- If the notification list page is active, we rebuild the screen (line 23).

For-in loop

The *for* statement can be used to declare a *for-in* loop.

Syntax:

```
for(var v in iterable) {
  segment
}
```

In the expression, an element will be picked from the iterable (e.g., List, Set) each time until the loop walks through all elements in the iterable.

Example:

```
for(var c in [‘H’, ‘E’, ‘L’, ‘L’, ‘O’]) {
  print(c);
}
```

The code above prints five lines, they are respectively ‘H’, ‘E’, ‘L’, ‘L’ and ‘O’.

4. Add the following **initState()** method to the **NotificationListState** class. It is used to invoke the methods we declared above.

```
1   @override
2   void initState() {
3     super.initState();
4     getRoles().then((_) => getNotificationList());
5   }
```

- Invoke the **getRoles()** method (line 4). Its callback function then will call the **getNotificationList()** for downloading the notifications.

5. Open the **notification_list.dart** file and modify the **build()** function of the **NotificationListState** class:

✪ *only the stared lines are changed.*

```
1       @override
2       Widget build(BuildContext context) {
3         var widgetList = <Widget>[];
4
5*        var data = List();
6*
7*        for (List c in nMap.values)
8*          data.addAll(c);
9*
10*       data.sort((a, b) => b['createdAt'] - a['createdAt']);
11*
12*   //    for (var i = 1; i <= 20; i++) {
13*   //      var item = 'Notification $i';
14*       for (var i=0; i<data.length; i++){
15*
16*         var item = data[i];
17*         var title = item['title'];
18*         var course = item['course'];
19*         var datetime = DateTime.fromMillisecondsSinceEpoch(item['createdAt']);
20*         var createdAt = DateFormat('EEE, MMMM d, y H:m:s',
21*           'en_US').format(datetime);
22*
23          widgetList.add(
24              ListTile(
25                leading: Icon(Icons.notifications),
26*   //            title: Text('Item $i'),
27*   //            trailing: Icon(Icons.face),
28*               title: Column(
29*                 crossAxisAlignment: CrossAxisAlignment.stretch,
30*                 children: <Widget>[
31*                   Text(title, style: TextStyle(fontWeight: FontWeight.bold),),
32*                   Text(createdAt,
33*                     style: TextStyle(fontSize: 10.0, color: Colors.blueGrey),),
34*                 ],
35*               ),
36*               trailing: Text(course.replaceAll(' ', '\n'),
37*                 textAlign: TextAlign.right,),
38
39                onTap: () {
40                  notificationSelection = item,
41                  Navigator.pushNamed(context, '/notificationView');
42                },
43              )
44          );
45      }
```

- Variable **data** is a list for storing notifications downloaded from the **Firebase** database (line 5). The *for-in* loop reads the values from **nMap**, and add them to **data** (line 7 ~ 8).

- We sort the notifications by creation time (the **createdAt** field) in descending order (latest first) using the **sort()** method with sorting description (an inline function to describe how to sort) (line 10).

- Then, we use the elements of the **data** list to create **ListTile** widgets. We also need notification's title, created time and course (line 16 ~ 21). The **createdAt** field stores a time-stamp, we convert it to date and time (line 19 ~ 21).

- Then, we put these things into a **ListTile** widget (line 28 ~ 37).

```
46        return Scaffold(
47          appBar: AppBar(title: Text('Notifications'),),
48          body: ListView(
49            children: widgetList,
50            padding: EdgeInsets.all(20.0),
51          ),
52
53*         floatingActionButton: (canCreate)?
54*         FloatingActionButton(
55*             child: Icon(Icons.add),
56*             onPressed: ()=>Navigator.pushNamed(context, '/notificationCreate'),
57*           ) : null,
58        );
59      }
60    }
```

✪ *only the stared lines are changed.*

- We show a float-action button if the **canCreate** equals true. The callback function of the float action button is used to build a new screen for creating a notification (line 53 ~ 57). But, the code of the notification creation screen is not yet implemented.

6. Save the file and re-run your app. You should see a public notification in the notification list page.

7. Open the **Firebase** console and add a new entry under the 'users' entry with your student ID and a student role for COMP 7510, same as the 'xxxxxxxxxx' entry.

8. Re-open the notification list page on the app. Now, you should see the notifications of COMP 7510 too.

9. Go to the **Firebase** console again and change your role to administrator or teacher for COMP 7510. And, re-open the notification list page on the app. You should now see a float action button at the bottom right corner.

Updating the Firebase Database

Updating data in the **Firebase** database is quite simple, we first use a database reference pointing to the position you want to update. Then, we simply call the methods – **push()** / **set()** / **remove()**, provided by the database reference to update the data.

Adding New Entry

The **push()** method creates a database reference object that points to a new position. The **set()** method is used to set the value for an existing entry. With these two methods, we can add a new entry to the database.

For example, we need to post a new notification to everyone. The details of the notification are as follows:

Field	Values
Title	System Maintenance
Content	REACH will not be available on Oct 1, 2018, 00:00 - 02:00.
Course	ALL

The following codes can be used to create the new notification:

```
1   var ref = dbRef.child('courses/ALL/notifications').push();
2
3   var key = ref.key;
4
5   ref.set({
6     'key' : key,
7     'course' : 'ALL',
8     'title' : 'System Maintenance',
9     'content' : 'REACH will not be available on Oct 1, 2018, 00:00 – 02:00',
10    'createdAt' : DateTime.now().millisecondsSinceEpoch,
11    'createdBy' : userID,
12  });
```

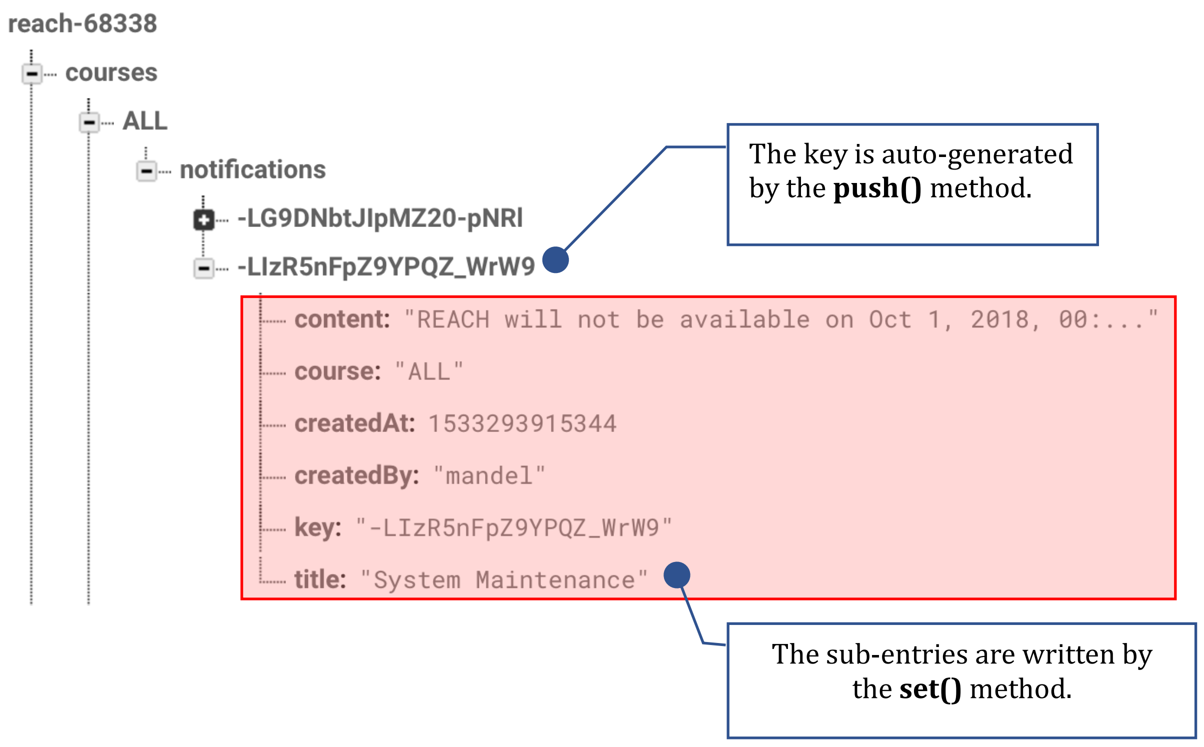
- The **push()** method creates a new empty entry under the position 'course/ALL/notification'. And, we assign the returned database reference to variable **ref** (line 1).

- A unique key is generated by the **push()** method and stored in the database reference (line 3).

- The **set()** method accepts a value (the map), and write it to the new entry (starting from line 5).

- The **DateTime.now()** method returns the current date and time (line 10). The **millisecondsSinceEpoch** property of a **DateTime** object outputs its time stamp.

The following is the example result in the **Firebase** database:


```
reach-68338
  courses
    ALL
      notifications
        -LG9DNbtJIpMZ20-pNRl
        -LIzR5nFpZ9YPQZ_WrW9
          content: "REACH will not be available on Oct 1, 2018, 00:..."
          course: "ALL"
          createdAt: 1533293915344
          createdBy: "mandel"
          key: "-LIzR5nFpZ9YPQZ_WrW9"
          title: "System Maintenance"
```
The key is auto-generated by the **push()** method.
The sub-entries are written by the **set()** method.

Changing Existing Entry

To update an existing entry, we only need to use the **set()** method to write the data to the specific position directly. For example, the maintenance date is not correct, it should be Oct 2, 2018. The following codes can be used to correct it:

```
var ref = dbRef.child(
    'courses/ALL/notifications/-LIzR5nFpZ9YPQZ_WrW9/content');

ref.set('REACH will not be available on Oct 2, 2018, 00:00 – 02:00');
```

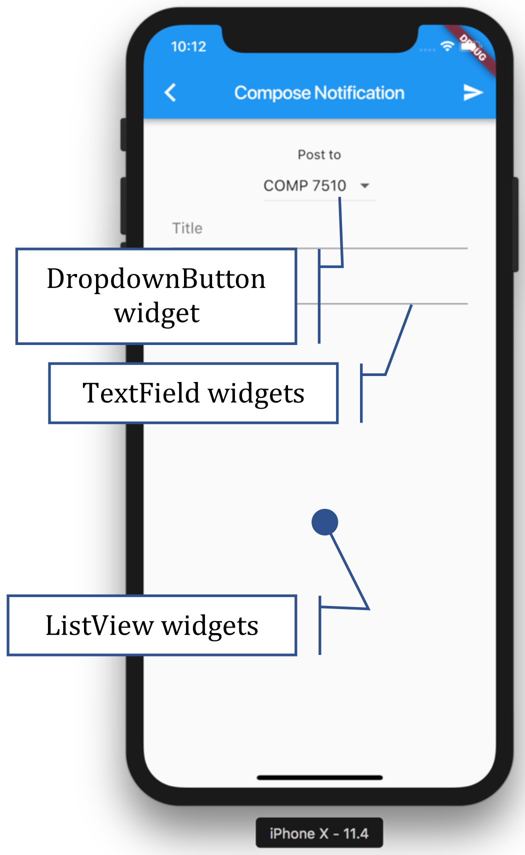
Removing Existing Entry

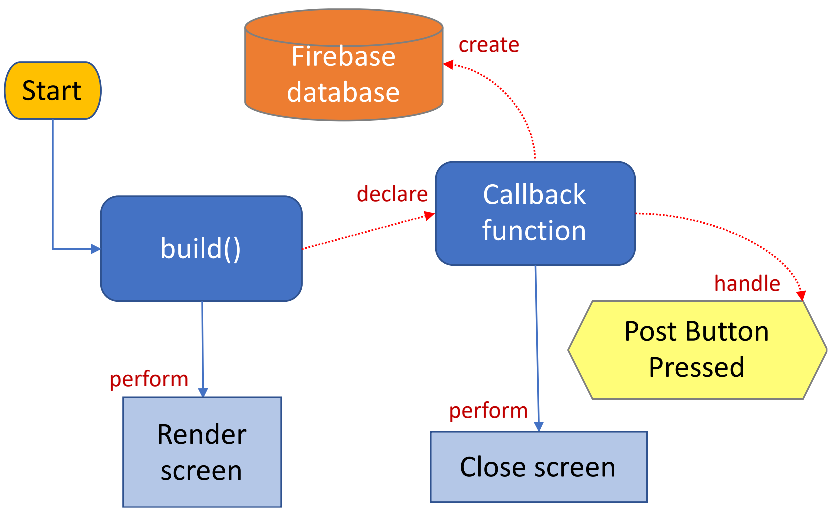
The deletion is very similar to updating an existing entry. We use the **remove()** method to delete the entry directly. For example, we need to delete the system maintenance notification because the system maintenance is canceled. The following codes can be used to delete the notification:

```
var ref = dbRef.child(
    'courses/ALL/notifications/-LIzR5nFpZ9YPQZ_WrW9/content');

ref.remove();
```

Creating Notification in Our App

The staff (teachers and administrators) working on a course are able to post notifications to the course. For example, I am the teacher of COMP7510, I can post notifications to COMP7510. Of course, we need to have a new user interface for creating notifications. Therefore, we are going to add a new user interface to our app.





The workflow of the new user interface is quite simple. The **build()** method builds the interface and declares a callback function to handle the *On-Pressed* event of the *Send* button. When the user presses the *Send* button after composing the notification, the callback function will be invoked. The callback function creates a new entry with the notification details in the **Firebase** database.

Widgets

Three new widgets are used in this user interface, the **DropdownButton**, **TextField** and **Divider** widgets.

DropdownButton

The **DropdownButton** widget provides a dropdown list for selecting items (**DropdownMenuItem** widgets). When the user changes the selection of the **DropdownButton** widget, its *On-Changed* callback function will be invoked. And, its **value** property then stores the selection.

TextField

The **TextField** widget is used for handling user's text input. When the user tries something in the **TextField** widget, its *On-Changed* callback function will be invoked. The callback function can be used for updating internal variables, input validation or so on. In addition, the **TextField** widget has many properties for customizing its input mode, such as:

- The **keyboardType** property specifies the keyboard type to multiple lines, email input, number input, etc.
- The **decoration** property shows the hint text when the text field is empty.
- The **maxLines** property specifies the maximum number of lines. The value equals null that indicates no limitation. Its default value is 1.

Adding Code

Let's follow the steps below to create a new user interface:

1. Create a new file named **notification_create.dart**, and add the following codes to the file:

```
 1  import 'global.dart';
 2  import 'package:flutter/material.dart';
 3
 4  class NotificationCreationPage extends StatefulWidget {
 5    @override createState() => NotificationCreationState();
 6  }
 7
 8  class NotificationCreationState extends State {
 9    var selectedCourse = roles.keys.first;
10    var title = '';
11    var content = '';
12
13    @override
14    Widget build(BuildContext context) {
15
16    }
17  }
```

- Two new classes are declared – **NotificationCreationPage** (line 4 ~ 6) and **NotificationCreationState** (starting from line 8).

2. Many things are needed to be done in the **build()** method. First, prepare a **DropdownButton** widget. Add the following code segments to the **build()** method:

```
1   var items = <DropdownMenuItem>[];
2
3   for (var k in roles.keys) {
4     var v = roles[k];
5
6     if (['teacher', 'administrator'].contains(v))
7       items.add(DropdownMenuItem(value: k, child: Text(k)));
8   }
9
10  var ddButton = DropdownButton(
11    value: selectedCourse,
12    items: items,
13    onChanged: (course) => setState(() => selectedCourse = course),
14  );
```

- we prepare a list of the **DropdownMenuItem** widgets. These menu items store the course codes that are retrieved from the global variable **roles**.

- The *for-in* loop gets the key (course code) and the value (user role) from the **roles** map (line 3 ~ 8). We check whether the user role equals to teacher or administrator. If yes, it indicates the user can post notifications for that course. So, we add a menu item with the course code to the **items** list.

- We create a **DropdownButton** widget that contains the recently declared menu items. The drop-down button is used for selecting a course (line 10 ~ 14).

3. Declare a list that stores different widgets including a **Text**, **DropdownButton**, **TextField** widgets. Add the codes below to the **build()** method following the previous code segment:

```
var widgets = <Widget>[
  Text('Post to'),

  ddButton,

  TextField(
    decoration: InputDecoration(hintText: 'Title',),
    onChanged: (text) => setState(() => title = text),
  ),

  Divider(color: Colors.transparent,),

  TextField(
    decoration: InputDecoration(hintText: 'Content',),
    keyboardType: TextInputType.multiline,
    maxLines: null,
    onChanged: (text) => setState(() => content = text),
  ),
];
```

- The widget list stores a **DropdownButton** widget and two **TextField** widgets separated by a **Divider** widget.

- The **TextField** widgets are used for the title and content input respectively.

4. Declare a scaffold with the widget and return it. Add the codes below to the **build()** method following the previous segment:

```
return Scaffold(
  appBar: AppBar(
    title: Text('Compose Notification'),
    actions: <Widget>[
      IconButton(icon: Icon(Icons.send), onPressed: () => post(),),
    ],
  ),
  body: ListView(
    padding: EdgeInsets.all(30.0),
    children: <Widget>[Column(children: widgets)],
  ),
);
```

- An **IconButton** widget is declared in the app bar. It is used for sending the notification to the **Firebase** database. The **post()** method will be invoked when the user presses the button.

5. Add the following **post()** method in the **NotificationCreationState** class:

```
1   void post() {
2     var ref = dbRef.child('courses/$selectedCourse/notifications').push();
3
4     ref.set({
5       'key' : ref.key,
6       'course' : selectedCourse,
7       'title' : title,
8       'content' : content,
9       'createdAt' : DateTime.now().millisecondsSinceEpoch,
10      'createdBy' : userID,
11    });
12
13    Navigator.pop(context);
14  }
```

- The **post()** method is used for adding a new notification entry to the **Firebase** database.

- The **push()** method provides a new position under the "courses/$selectedCourse/notification" path (line 2). The **set()** method writes the details of the notification to the **Firebase** database, the position **ref** (line 4 ~ 11).

- The **Navigator.pop()** statement is used to close the notification create page (line 13).

6. Open the **main.dart** file and add an import statement for the **notification_create.dart** file.

```
import 'package:flutter/material.dart';
```

7. Update the **build()** method of the **MyApp** class as follows:

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: HomePage(),
    routes: <String, WidgetBuilder>{
      '/notificationList':
          (BuildContext context) => NotificationListPage(),

      '/notificationView':
          (BuildContext context) => NotificationViewPage(),

      '/notificationCreate':
          (BuildContext context) => NotificationCreationPage(),
    },
  );
}
```

8. Save the files and re-run the app.

9. Go to **Firebase** console, add an *administrator* role of COMP 7510 to your user entry.

10. Go to the iOS simulator, re-open the notification list page to refresh your roles. Then, press the "+" button to create a new notification.

11. Select "COMP 7510" using the drop-down menu button.

12. Input something for the title and content.

13. Press the *Send* icon button. The notification creation page should be closed then.

14. Check the notification list, there should be a new notification about the system maintenance.

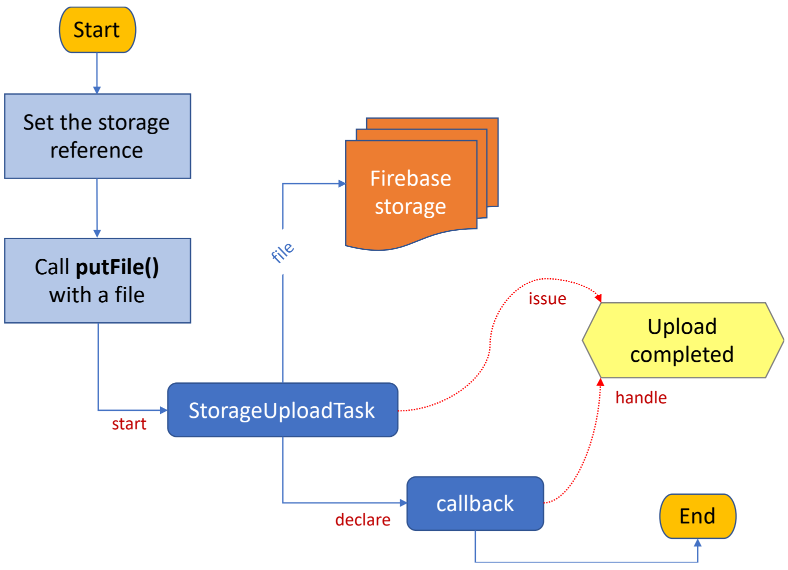
Uploading Files to Firebase Storage

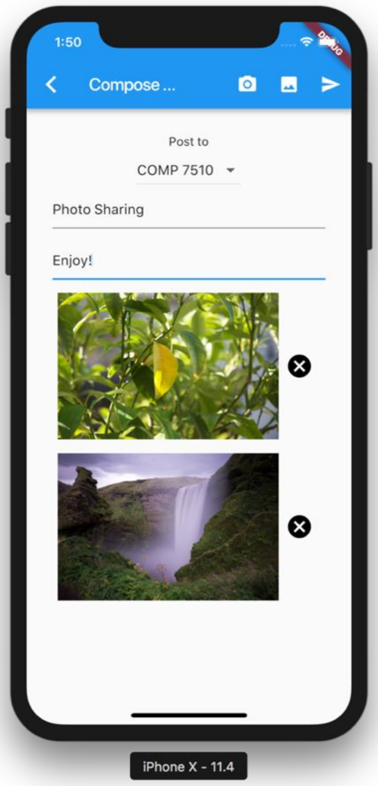
The **StorageReference.putFile()** method provided by **Firebase** API is used to upload a file to the **Firebase** storage. We use a **StorageReference** object to refer to a destination (position in the **Firebase** storage), and use the **putFile()** method to start a background task (**StorageUploadTask**) for uploading a file. With the **StorageUploadTask.future.then()** method, we declare a callback function for handling the upload completed event.

Here is an example for uploading a jpeg file:

```
var fRef = storageRef.child('images/img01.jpg');

var task = fRef.putFile('/~/Documents/myPhoto.jpg');

task.future.then((snapshot) => print(snapshot.downloadUrl.toString()));
```

- Declare the destination using the StorageReference object.
- Start uploading the source file to the destination.
- Declare a callback function to handle the upload completed event.
- Print the download URL of the uploaded file.





Uploading Photos in Our App

We are going to improve the notification creation function so as to allow attaching photos in the notifications. Two buttons will be added in the app bar for picking a photo from the photo gallery or the camera. The photos will be shown immediately with cancel buttons. When the user presses the send button, the photos will be uploaded to the **Firebase** storage, the notification with the photo download URLs will be written in the **Firebase** database.

Let's follow the steps below:

1. Open the **notification_create.dart** file and add a new list in the **NotificationCreationState** class for storing image files:

```
     class NotificationCreationState extends State {
 ➔     var images = [];
       …
     }
```

2. Add the following import statement at the top of the file:

```
import 'package:image_picker/image_picker.dart';
```

3. Add the **attach()** method to the **NotificationCreationState** class:

```
 1   void attach(source) {
 2     ImagePicker.pickImage(source: source).then((file){
 3       if (file != null)
 4         setState(() => images.add(file));
 5     });
 6   }
```

- The **ImagePicker.pickImage()** method launches the *Camera* app or *Gallery* app for picking a photo (line 2). If the user cancels the image picking, variable file equals null (line 3). Otherwise, variable file refers to the selected photo.

- In the callback function, the selected photo file will be added to the **images** list (line 4).

4. In the **build()** method of the **NotificationCreationState** class, we add two icon buttons to the app bar:

```
     return Scaffold(
       appBar: AppBar(
         title: Text('Compose Notification'),

         actions: <Widget>[
 ➔         IconButton(
 ➔           icon: Icon(Icons.camera_alt),
 ➔           onPressed: () => attach(ImageSource.camera),
 ➔         ),
 ➔         IconButton(
 ➔           icon: Icon(Icons.photo),
 ➔           onPressed: () => attach(ImageSource.gallery),
 ➔         ),
           IconButton(
             icon: Icon(Icons.send),
             onPressed: () => post(),
           ),
         ],
       ),
       ...
     );
```

5. The **attach()** method will be invoked when the On-Pressed event occurs. **ImageSource.camera** indicates “photo from camera”; **ImageSource.gallery** indicates “photo from gallery”.

6. To show the attached photos in the screen, add the following code segments to the **build()** method before returning the scaffold:

```
1   var width = MediaQuery.of(context).size.width - 120;
2
3   for (var f in images) {
4     widgets.add(Divider(color: Colors.transparent,));
5
6     widgets.add(
7
8       Row(
9         mainAxisAlignment: MainAxisAlignment.center,
10        children: <Widget>[
11
12          Image.file(f, width: width),
13
14          IconButton(icon: Icon(Icons.cancel), iconSize: 32.0,
15            onPressed: () => setState(() => images.remove(f)),
16          )
17
18        ],
19      )
20    );
21  }
```

- In the *for-in* loop, we retrieve the image files from the images list.

- For each image, we show a transparent divider, the image, and an icon button. These widgets are added to the **widgets** list (line 4 ~ 20).

- The **image.file()** method creates an Image widget using the image file (line 12).

- The icon buttons are used to remove the corresponding images from the **images** list (line 15).

7. To upload the attached photos to the **Firebase** storage, add the new code segments to the **post()** method before closing the screen:

```
1    void post() {
2      var ref = dbRef.child('courses/$selectedCourse/notifications').push();
3      var key = ref.key;
4
5      ref.set({
6        'key' : key,
7        'course' : selectedCourse,
8        'title' : title,
9        'content' : content,
10       'createdAt' : DateTime.now().millisecondsSinceEpoch,
11       'createdBy' : userID,
12     });
13
14*    for (var i=0; i < images.length; i++) {
15*      var fRef = storageRef.child(ref.key + '/$i');
16*      var task = fRef.putFile(images[i]);
17*      task.future.then((snapshot) =>
18*        ref.child('images/$i').set(snapshot.downloadUrl.toString())
19*      );
20*    }
21
22     Navigator.pop(context);
23   }
```

- In the *for-in* loop, we retrieve the images from the images list. For each image, we declare a storage reference **fRef** refers to the destination path (line 15). The **putFile()** method starts a upload task (line 18).

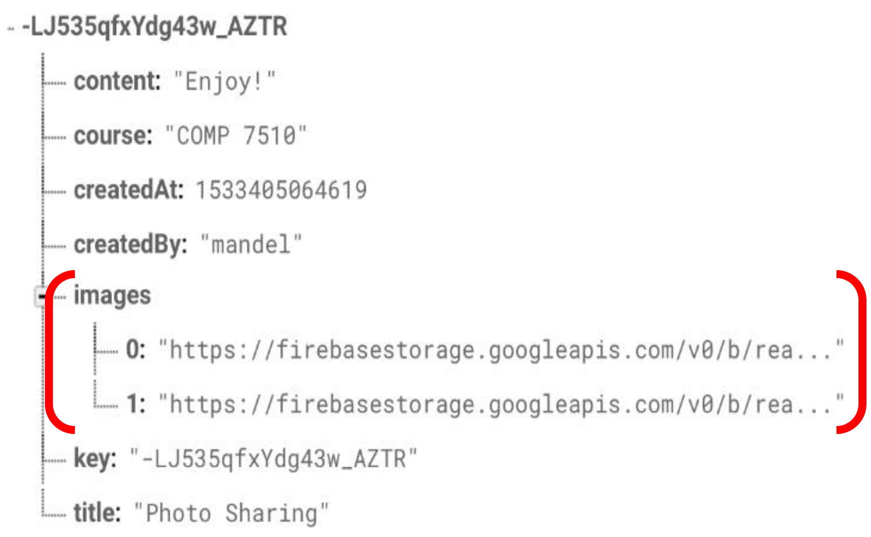
- In the callback function, we retrieve the download URL of the uploaded file and store the URL to the notification entry (line 18).

8. Open the **/reach/ios/Runner/Info.plist** file, add the following keys in the **dict** tag:

```
<key>NSPhotoLibraryUsageDescription</key>
<string>This app requires access to the photo library.</string>
<key>NSCameraUsageDescription</key>
<string>This app requires access to the camera.</string>
```

9. Save the files and re-run your app.

10. Create a new notification. Use the *camera* button or *gallery* button to add some photos. Press the *cancel* button next to the photo to remove it. Then, press the *send* button to send the notification.



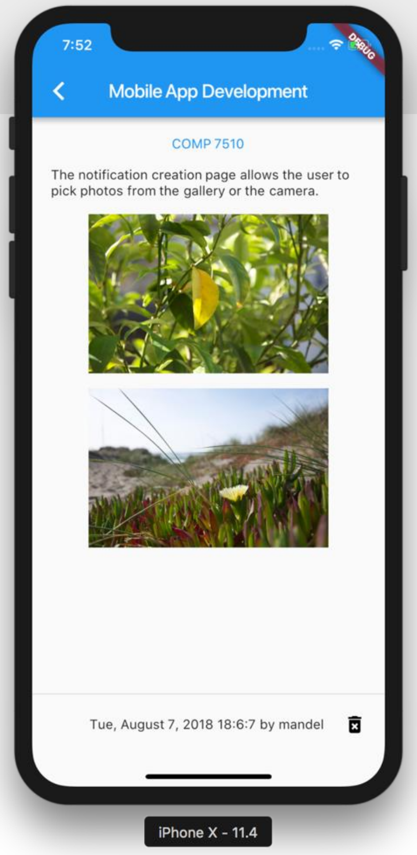
```
- -LJ535qfxYdg43w_AZTR
    content: "Enjoy!"
    course: "COMP 7510"
    createdAt: 1533405064619
    createdBy: "mandel"
  - images
      0: "https://firebasestorage.googleapis.com/v0/b/rea..."
      1: "https://firebasestorage.googleapis.com/v0/b/rea..."
    key: "-LJ535qfxYdg43w_AZTR"
    title: "Photo Sharing"
```

Viewing the Notification Details

Now, we are going to list the user interface – notification view page to the **Firebase** database. Remember that the item will be stored in the global variable **notificationSelection** when the user taps on an item in the list of the notification list page. The item stores the notification details, and we show them in the notification view page.

Let's rewrite the codes of the **notification_view.dart** as follows:

```dart
import 'global.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:async';
import 'package:intl/intl.dart';

class NotificationViewPage extends StatefulWidget {
  @override createState() => NotificationViewState();
}

class NotificationViewState extends State {

  var images = [];

  @override
  void initState() {
    super.initState();

    if (notificationSelection['images'] != null)
      for (var url in notificationSelection['images']) {
        http.get(url).then((response){
          images.add(response.bodyBytes);

          if (mounted)
            setState((){});
        });
      }
  }

```


7:52
DEBUG
Mobile App Development
COMP 7510
The notification creation page allows the user to pick photos from the gallery or the camera.
Tue, August 7, 2018 18:6:7 by mandel
iPhone X - 11.4

```
  @override
  Widget build(BuildContext context) {
    var data = notificationSelection;
    var title = data['title'];
    var course = data['course'];
    var content = data['content'];
    var createdBy = data['createdBy'];
    var datetime = DateTime.fromMillisecondsSinceEpoch(data['createdAt']);
    var createdAt = DateFormat('EEE, MMMM d, y H:m:s', 'en_US').format(datetime);

    var childWidgets = <Widget>[
        Text(course, style: TextStyle(color: Colors.blue),),
        Divider(color: Colors.transparent,),
        Text(content),
      ];

    var width = MediaQuery.of(context).size.width - 120;

    for (var i in images) {
      childWidgets.add(Divider(color: Colors.transparent));
      childWidgets.add(Image.memory(i, width: width));
    }

    return Scaffold(
      appBar: AppBar(
        title: Text(title),
      ),
      body: ListView(
        padding: EdgeInsets.all(20.0),
        children: <Widget>[
          Column(
            children: childWidgets,
          ),
        ],
      ),

      persistentFooterButtons: <Widget>[
        Text('$createdAt by $createdBy'),
        (['teacher', 'administrator'].contains(roles[course]))?
        IconButton(
          icon: Icon(Icons.delete_forever),
          onPressed: () => delete(),
        ):null,
      ],
    );
  }

  void delete() {
    var key = notificationSelection['key'];
    var course = notificationSelection['course'];

    dbRef.child('courses/$course/notifications/$key').remove();

    for (var i = 0; i < images.length; i++)
      storageRef.child('$key/$i').delete();

    Navigator.pop(context);
  }
}
```

- Before building the user interface, send http request to the **Firebase** storage for downloading image files (line 19 ~ 27). If the file download completes, check whether the user interface is still available or not before invoking the **setState()** method.

- Variable **data** stores the current notification, exactly the same as the global variable **notificationSelection** (line 30). It is just used to make the variable name shorter.

- Read the title, course, content, createdBy, and createAt from the notification (line 32 ~ 38). Then, use the information to create widgets and store them in the widget list (line 40 ~ 44).

- If the images list stores image data, use the data to create Image widgets (line 48 ~ 51).

- An icon button is added in the footer for deleting the notification. The icon button will be shown if the user is the teacher or administrator of the course the notification belongs to (line 69 ~ 72).

- A new method **delete()** is added. It is the callback method for deleting the current notification from the **Firebase** database and the corresponding image files from the **Firebase** storage (line 77 ~ 87).

Exercise

Currently, your app has the functions about notifications only. For the course project, you need to add additional functions, such as group formation, chatroom, forum, appointment management, administration tool, etc.

Your tasks are:

1. Discuss with your team members what additional functions should be added.

2. Design the work flows and user interfaces of the new functions.

3. Change the existing screens to match your design

Appendix – Running the App in iPhone / iPad

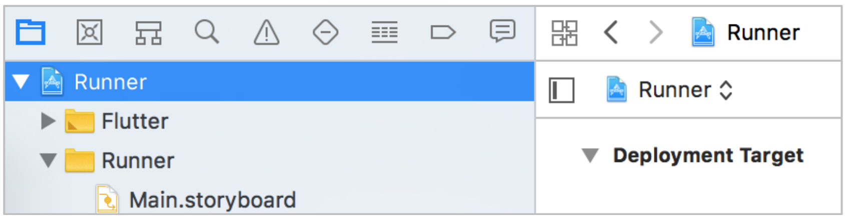
Currently, we use the simulator to run our app. If you want to test your app in your iPhone or iPad, you can follow the following steps:

1. In IntelliJ, save all files.

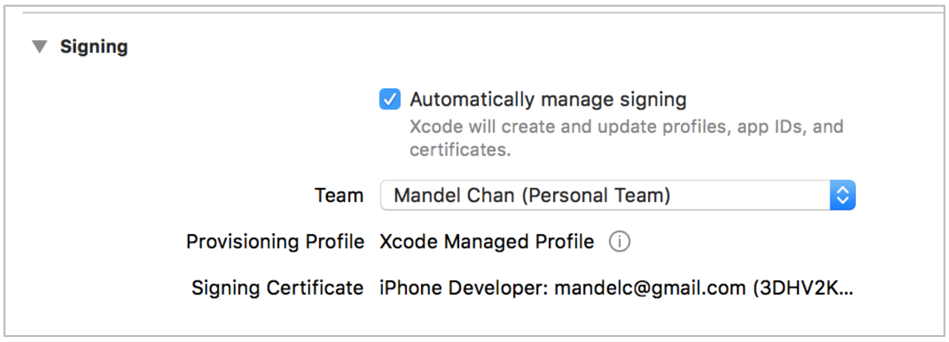
2. Use Finder to open "~/Documents/reach/ios/Runner.xcworkspace.

3. In Xcode, select the menu "Xcode" > "Preference…". Select the "Account" tab and add your apple ID.

4. In Project Navigator, select the project root – "Runner". Then, change project: Runner to target: Runner.



5. In the signing area, change the team to your personal team.



Signing

☑ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: Mandel Chan (Personal Team)
Provisioning Profile: Xcode Managed Profile
Signing Certificate: iPhone Developer: mandelc@gmail.com (3DHV2K…

6. Select the menu "Product" > "Destination" > your device.

7. Click the run button of Xcode to run your app.

8. Type the Mac login password and click "Always Allow" if it prompts you about the password for the keychain.

References

[1] Display images from the internet. (n.d.). Retrieved from https://flutter.io/cookbook/images/network-image/

[2] Firebase for Flutter. (n.d.). Retrieved from https://codelabs.developers.google.com/codelabs/flutter-firebase/#0

[3] firebase_database | Flutter Package. (n.d.). Retrieved from https://pub.dartlang.org/packages/firebase_database

[4] firebase_storage | Flutter Package. (n.d.). Retrieved from https://pub.dartlang.org/packages/firebase_storage

[5] Flutter - Beautiful native apps in record time. (n.d.). Retrieved from https://flutter.io/

[6] A Tour of the Dart Language. (n.d.). Retrieved from https://www.dartlang.org/guides/language/language-tour