Code Quality Review Report

NG911

Team Members:
Rabaa Abdelrahman,
Li Pan,
Renz Rivero
April 9, 2022

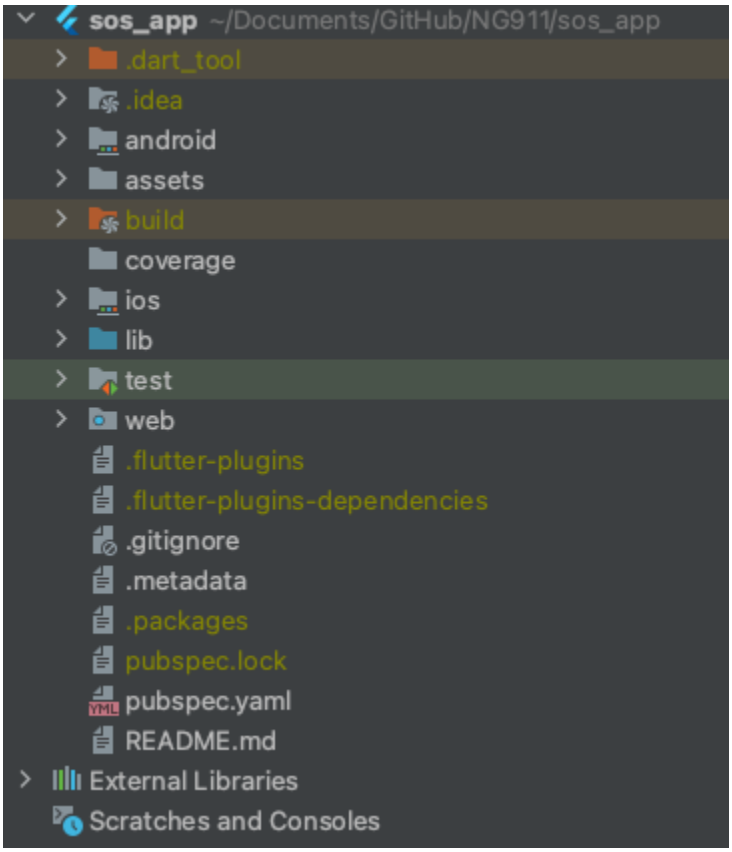
Table of Content

Code Formatting

NG911 includes one mobile SOS application and one website PSAP service application. For both applications we used Android Studio IDE standard to format our code. First of all, our code aligns properly and with proper spacing. It makes it so that each section of our code is unique. Secondly, the names of parameters, classes, and files also have proper format. Similarly, it is easy to find the files or functions needed. Certainly, our code window fixes the standard laptop screen, and it is unnecessary for a horizontal scroll to view it.

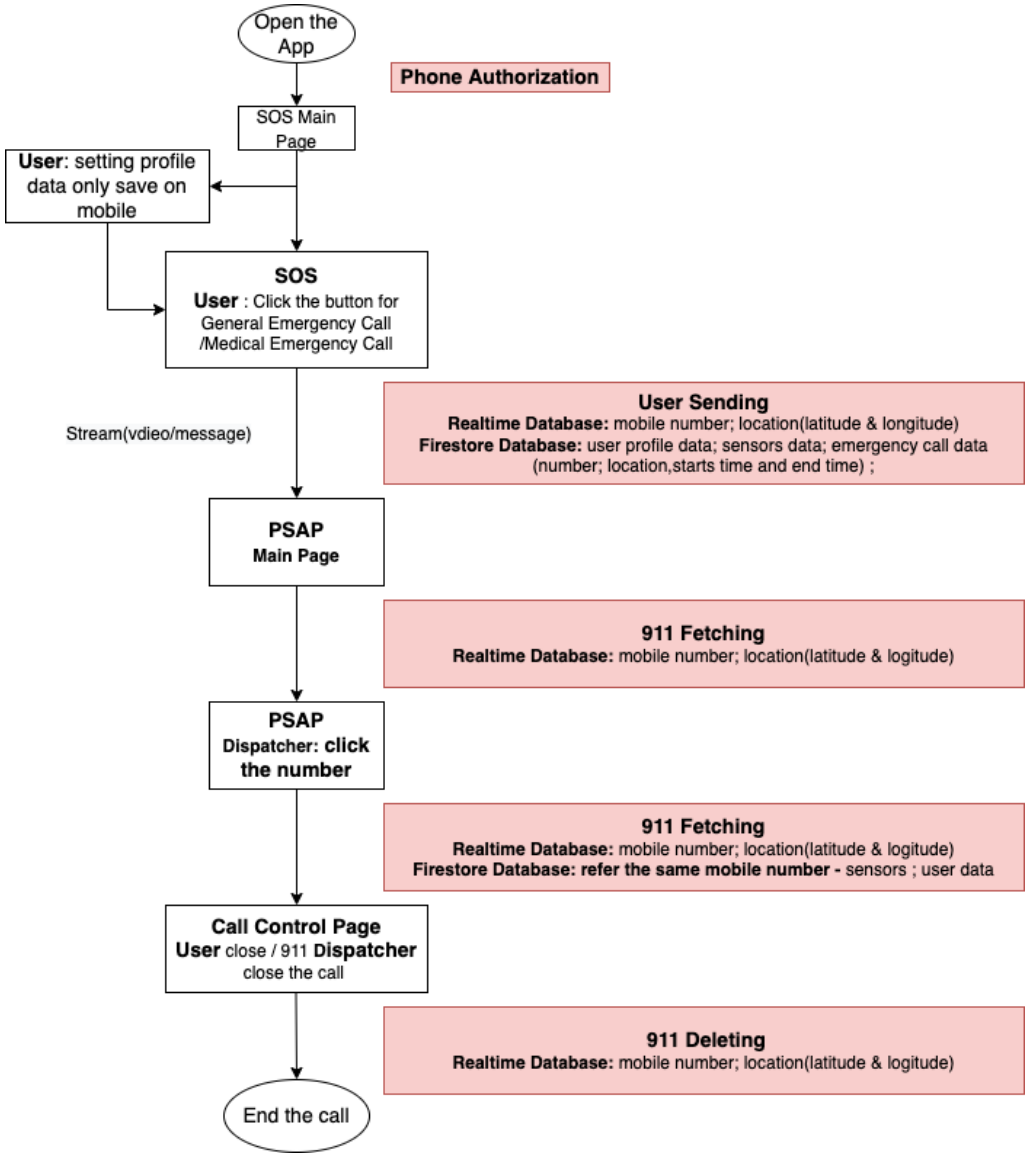
Architecture

The SOS application is compatible with iOS and Android, all configured files have separate locations at different folders for each operating system. The library code files hierarchy is following our project architecture. It can be viewed through different folders.



The user interface pages and server files are located in separate folders, that means the project widgets and functions are separate. This design makes the code easy to edit. Our project is developed mostly using Dart object-oriented language. We are following Dart syntax coding. Similarly, we have html files used for the PSAP application that uses HTML syntax. The PSAP side design pattern is factory pattern. Each emergency uses the same code, but the information referring to the emergency is unique.

Also, our code functions architecture is following data flow technique. The functions are called by the order of the data transmit requirement. For example, when a user makes an emergency call, the SOS application calls the function to send the data of the user's mobile information only if the user did not add any information under their profile.



Coding best practices

There is no hard code in our project. It only has constant value and configuration value. All variables are fetched through the functions or databases. For example, the mobile acceleration data is fetched through the mobile accelerometer sensor, then streamed into our real-time database. Similarly, the geolocation data is fetched through the function of fetching GPS data.

```
// Acceleration Data
databaseReal.child('Online').onValue.listen((event) async {
  bool onlineB = event.snapshot.value as bool;
  online = onlineB;
  if (online! == true && ended != true) {
    _stopWatchTimer.onExecute.add(StopWatchExecute.stop); //Stop timer

    streamSubscription = accelerationC.stream.listen((event) {
      accelerationString = event;
      if (ended != true) {
        databaseReal.update({
          'Acceleration': accelerationString,
        });
      }
    });

    // Location Data
    Location location = Location();
    await location.getCurrentLocation();
    streamLocationSubscription = stream.listen((DatabaseEvent event) async {
      if (ended != true) {
        databaseReal.update({
          'Latitude': location.latitude.toString(),
          'Longitude': location.longitude.toString(),
          'Speed': location.speed.toString()
        });
      }
    });
```

In our project we group similar values under enumeration. For example, the users need to sign up before using the mobile application. Our project implements firebase mobile authorization function which requires the user to enter their mobile number and OTP code. It includes two stages of the users sign up process. We group these two states in a numeration. The following image shows the state we keep.

```
enum MobileVerificationState {
  SHOW_MOBILE_FORM_STATE,
  SHOW_OTP_FORM_STATE,
}
```

Besides Flutter's default widget comments, we added reasonable comments for our functions. These comments specify the purpose of each function. It will be easy to maintain our project in the future. In addition, we don't use multiple if or else blocks, instead we use the switch function. It gives multiple possibilities for function approaches. At the same time, there is more space for further project development. It is easy to add more cases instead of using logic inside the if or else condition.

```
switch (type) {
  case 1: // Personal call, only first permission is granted
    return Column(children: [
      const Text("Emergency contact number: "),
      Text(emergencyContactNumberString),
    ]);  // Column

  case 2: // Personal call, two permissions are granted

    return Column(children: [
      const Text("Emergency contact number: "),
      Text(emergencyContactNumberString),
      Text("Personal health card: " + personalHealthCardString),
      ElevatedButton(
          onPressed: downloadTheURL,
          child: const Text("Download Personal Medical Report")),  // ElevatedButton
    ]);  // Column

  case 3: // Personal call, only Medial permission is granted
    return Column(children: [
      Text("Personal health card: " + personalHealthCardString),
      ElevatedButton(
          onPressed: downloadTheURL,
          child: const Text("Download Personal Medical Report")),  // ElevatedButton
    ]);  // Column

  case 4: // Emergency contact call, with Permission granted
    return Column(children: [
      const Text("Emergency contact health card number: "),
      Text(emergencyHealthCardNumberString),
      ElevatedButton(
          onPressed: downloadTheURL,
          child: const Text("Download Emergency contact Medical Report")),  // ElevatedButton
    ]);  // Column
```

Non Functional requirements

1.maintainability

The code is maintainable, as it has a clear hierarchy so the files are organized properly. The variables, functions and classes are all without ambiguous names. As such, It will be easy to identify and modify these within the later maintenance process. Our XML file keeps configures and lists all the plugins we are using for each project. However, we don't have any automated testing code, but we did many user case testing. Besides, for the SOS application we tested on different mobile operating systems and versions.

2. Reusability

The code is reusable because our functions and classes can be reused. These functions and classes are independent, which can be used at different files we need. It reduces the duplicate code and also simply our project. Similarly, these functions can be used for other projects requiring a similar goal. Our code is extensible because it uses minimal code changes to extend functionality for our project.

3. Reliability

The code is reliable; we catch exceptions and dispose of unexpected results. The following image shows how we handle different situations when a user enters information to the application.

```
//send user eHealth card number with permission
if (user.personalMedicalPermission == true) {
  await users
      .doc(user.mobile)
      .collection('Profile Information')
      .doc('Medical Info')
      .set({
        'medical permission': user.personalMedicalPermission,
        'personal health card': user.personalHealthNum,
        'personal medical file': user.personalMedicalFile,
      })
      .then((value) => print("User Updated"))
      .catchError((error) => print("Failed to add user: $error"));
```

4. Security

Our code is secure. By using authentication, that SOS application verifies users' mobile identity. The data transfer between SOS application and PSAP application is done by using the RSA decryption method. In addition, the PSAP login is implemented by hashing function. It ensures all our data is secure.

```
import 'package:pointycastle/api.dart' as crypto;
import 'package:rsa_encrypt/rsa_encrypt.dart';

Future<crypto.AsymmetricKeyPair<crypto.PublicKey, crypto.PrivateKey>>
    getKeyPair() {
  var helper = RsaKeyHelper();
  return helper.computeRSAKeyPair(helper.getSecureRandom());
}
```

5. Performance

We use Github workflow functionality to check if each version of code is built successfully. To ensure our project has good performance, our PSAP website application uses a couple timers to refetch the data. Besides this, using a real time database ensures the updated data is downloaded directly to the application. By using asynchronous programming, which provides future, async and await functions makes our applications processing faster. Also, the Flutter provides both stateless and stateful widgets. A stateless widget's content never changes, and a stateful widget is dynamic. Implementing a proper widget increases our project's performance as well.

6. Scalability

The project is scalable. Our project allows an amount of users to interact with the database at the same time. Google Firebase especially can handle big data processing.

7. Usability

In addition, our code is usable. There are multiple functions that can be used for other projects which are related to similar functionality. For example, this following function can be used for uploading files to Firebase firestore.

```dart
import 'dart:io';
import 'dart:typed_data';
import 'package:firebase_storage/firebase_storage.dart';

class FirebaseApi {
  static UploadTask? uploadFile(String destination, File file) {
    try {
      final ref = FirebaseStorage.instance.ref(destination);
      return ref.putFile(file);
    } on FirebaseException catch (e) {
      return null;
    }
  }

  static UploadTask? uploadBytes(String destination, Uint8List data) {
    try {
      final ref = FirebaseStorage.instance.ref(destination);
      return ref.putData(data);
    } on FirebaseException catch (e) {
      return null;
    }
  }
}
```

Object-Oriented Analysis and Design Principles

1. Single Responsibility Principle (SRP)

First of all, our code satisfies the SRP principle. Dart is an object oriented language. We code each class by representing a single responsibility principle (SRP). One class only takes responsibility for one single responsibility. Similarly, one function only achieves one task.

2. Open Closed Principle

Secondly, We use the open closed principle. If the project needs new functionality, the existing code does not need to be modified, instead a new function or class should be added. Our project has low threshold and high ceiling, we can implement more functionality to our applications.

3. Interface Segregation

Third, we use multiple smaller widgets built together for the user interface. Flutter allows multiple layers of widgets wrapped together. To enable the interface segregation we don’t have dependencies involved in the interface.

Reference

https://www.evoketechnologies.com/blog/code-review-checklist-perform-effective-code-reviews

https://dart.dev/codelabs/async-await

https://docs.flutter.dev/development/ui/interactive#:~:text=A%20widget%20is%20either%20stateful,are%20examples%20of%20stateless%20widgets.