A CROSS-PLATFORM APPLICATION FOR COVID-19 DIAGNOSTIC

Hamza Chehili[1] and Mustapha Bensaada[2]

[1]Frères Mentouri University - Constantine 1,
LIRE Laboratory - Constantine 2, Algeria
[2]Abbas Laghrour University, Khenchela, Algeria

ABSTRACT

The emergency of the Covid 19 pandemic has led technology to seek solutions to the different problems caused by the disease. In the monitoring area, connected devices offer new possibilities to a rapid detection and intervention of the new cases. They allow remote diagnostic to infected patients with covid 19 symptoms. However, the heterogeneity of the platform requires applications' developers to develop specific solutions for each platform. In this paper, we propose a cross-platform application that permits developer to use one code to build applications in different platforms. The paper describes the architecture of the application by presenting its three parts: interface screens (UI), data manipulation and authentication implementation. Finally, we show selected screens of an android release as an example.

KEYWORDS

Cross-Platform, Covid 19, Application, Development.

1. INTRODUCTION

Clinical management requires typical Diagnosis and monitoring of diseases. In fact, the control of infectious diseases is really challenging because it needs both early detection and treatment, surveillance and outbreak, due to their rapid transmission to others [2]. It is also demanding regarding the quality of care it requires and the limited setting offered. Indeed, the control of this kind of disease is confronted by logistical, educational, and temporal constraints that are intensified by a high volume of patients[4]. Therefore, a patient should take long distances in order to visit various clinics to be able to receive the appropriate treatment. This delays prescribing of treatment with amplified risk of suffering, mortality, and also inaccurate prescription of antimicrobials[3].

In this context, technology has brought various solutions to overcome those difficulties. new possibilities have emerged as possible additions to the traditional public health infrastructure [1, 5]. rapidity, simplicity, and probability are best illustrations for the most wanted features of tests and devices intended for the diagnostic process at the point of care (POC) [6, 14]. Indeed, digital health technology can help facilitating pandemic strategy and response in ways that are hard to realize manually [12]. In fact, having an easy accessibility to testing helps provides an easier treatment for patients. [2]. POC detection should be connected to front-end technology, such as smartphones, the Internet of Things (IoT) and cloud computing[7, 8].

Recently, the appearance of coronavirus pandemic (COVID-19) has led the world to a state of a challenging emergency. Broad efforts in testing for coronavirus infection, along with isolating

David C. Wyld et al. (Eds): AIAP, SIGML, CNSA, NIAI, CCSEIT - 2021
pp. 195-202, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.110113

infected cases and quarantining those in contact, have proven successful in managing the pandemic. Thus, a quick reaction towards this disease is important [10].
The rapid spread of the epidemic requires effective ways of detecting patients and diagnosing them to reduce the infection. [9]. Many aspects of technology are currently considered in the struggle against covid-19 among which is medical image processing, disease tracking, prediction outcomes, computational biology and medicines [11, 13].

This paper presents a cross-platform application for disease tracking by remote diagnostic of potential patients that have symptoms similar to those of covid 19 disease. This application is developed on Google Flutter framework. It allows code reuse across Android, IOS, Desktop and the Web [15]. This latter is in Betta version.

The paper's sections are organized as follows : section 2 gives the application description; section 3 presents the application build and release; and the conclusion refers to perspectives and future studies.

2. APPLICATION DESCRIPTION

The application is a layer on the flutter framework. Figure 1 gives a schematic view of the application files that contains the code used for user interface screens (UI), data manipulation and authentication implementation.

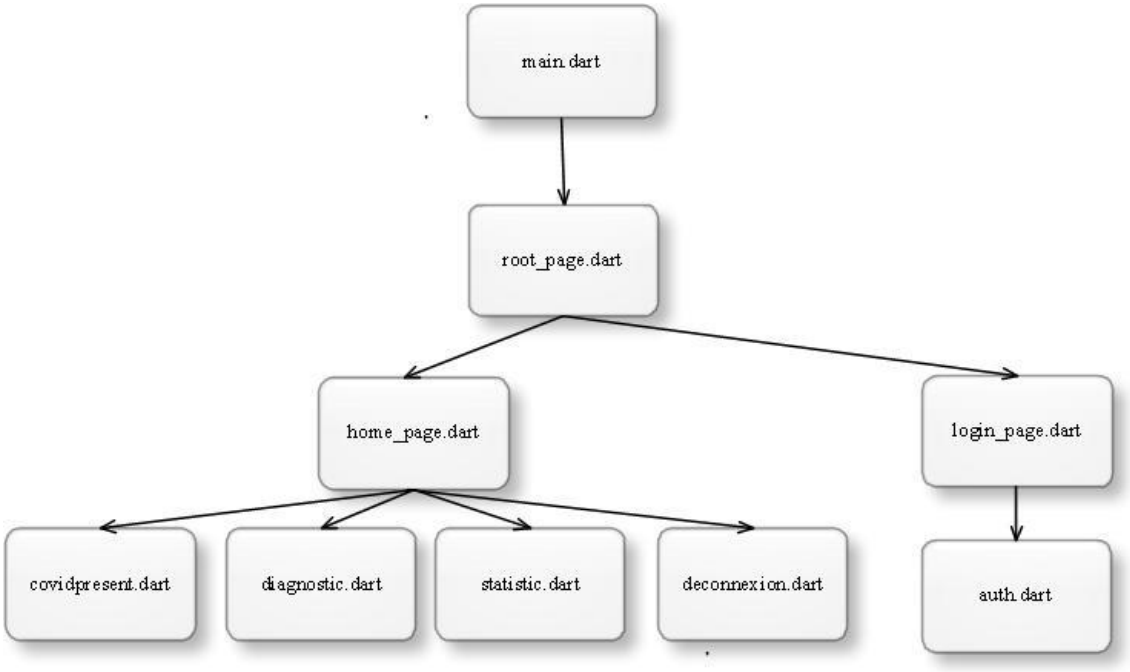

Figure 1. The application flux

2.1. User interface

Flutter applications are based on the Widget notion to design user interfaces. In our application, the widgets are implemented in the different files. The listing 1 shows the code of the build method of the class _LoginPageState that returns a Widget for the authentication interface.

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(...), // AppBar
    backgroundColor: Colors.grey[300],
    body: new SingleChildScrollView(child: new Container(
        padding: const EdgeInsets.all(16.0),
      child: new Column(
          children: [
        new Card(
          child: new Column(
              mainAxisSize: MainAxisSize.min,
              children: <Widget>[
            new Container(
                padding: const EdgeInsets.all(16.0),
              child: new Form(
                  key: formKey,
                child: new Column(
                  crossAxisAlignment: CrossAxisAlignment.stretch,
                  children: form() + submitWidgets(),
                ) // Column
              ) // Form
            ), // Container
          ]) // <Widget>[], Column
        ), // Card
        hintText()]))); // Column, Container, SingleChildScrollView, Scaffold
}
```

Listing 1. The build method of the class _LoginPageState

The listing 2 shows the code of the build method of the class _MyHomePageState that returns a Widget for the home interface.

```
@override
Widget build(BuildContext context) {
  int _act = 1;
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
      backgroundColor: Colors.green,), // AppBar
    drawer: Drawer(...), // Drawer
    body: Container(
      child: SingleChildScrollView(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          mainAxisSize: MainAxisSize.min,
          children: <Widget>[
            Container(
              child: Card(...),), // Container, InkWell, Card, Container
            Card(
              color: Colors.green,
              child: new InkWell(
                onTap: () {
                  Navigator.pushNamed(context, '/diagnostic');},
                child: Container(...),), // Container, InkWell
              ), // Card
            Card(...), // Card
            Card(...),], // Card, <Widget>[]
          ),),), // Column, SingleChildScrollView, Container
    ); // Scaffold
```

Listing 2. The build method of the class _MyHomePageState

The listing 3 shows the skeleton of the class _fomulaireglobalState that implements the diagnostic form and the result interface.

```
class _fomulaireglobalState extends State<fomulaireglobal> {
  Hypertension Hypertensindiag=Hypertension.pasHypertension;
  bool complete = false;
  bool repondu= false;
  int currentStep = 0;
  static final formKey_ig = new GlobalKey<FormState>();
  static final formKey_sym = new GlobalKey<FormState>();
  static final formKey_ant = new GlobalKey<FormState>();
  List formKeygloval=[formKey_ig, formKey_sym, formKey_ant];
  List<String> _authHint=['','',''];
  List<Step> formsteps(){...}
  @override
  Widget build(BuildContext context) {
   if (!repondu) {
     return Scaffold(
       appBar: AppBar(...), // AppBar
       body: Column(...), // Column
     ); // Scaffold
   }else {
     var resultat= score();
     return Scaffold(...);} // Scaffold
   }
    List<Widget> Informations_generales() {...}
    List<Widget> Symptomes() {...}
    List<Widget> Antecedent_medicaux() {...}
    Widget Boutton_retourner(VoidCallback onStepCancelm) {...}
```

Listing 3. The skeleton of the class _fomulaireglobalState

2.2. Data manipulation

The application data is composed of the diagnostic questions and the weighting of possible answers. This data is holed in JSON format and manipulated in the class calculate to calculate the diagnostic score. The listing 4 shows the method score that returns the diagnostic score.

```
double score(){
  double somme=0;
  for (reponse in reponses.keys) {
    if (cles.contains(reponse) && !(champsBool.contains(reponse))) {
      somme +=  bareme[reponse][reponses[reponse]];
      print(bareme[reponse][reponses[reponse]].runtimeType);
      print(bareme[reponse][reponses[reponse]]);

    }
    if (cles.contains(reponse) && champsBool.contains(reponse)) {
      somme += bareme[reponse][reponses[reponse].toString()];
    }
  }
  return somme;
}
```

Listing 4. The method score

2.3. Authentication implementation

This application uses the Firebase Authentication service to manage user Authentication (Sign in, Create new account, sign out and check the current user). The listing 5 shows the code of the file auth.dart that implements the code for Firebase Authentication services calling.

```
import 'dart:async';
import 'package:firebase_auth/firebase_auth.dart';
abstract class BaseAuth {
  Future<String> currentUser();
  Future<String> signIn(String email, String password);
  Future<String> createUser(String email, String password);
  Future<void> signOut();
}
class Auth implements BaseAuth {
  final FirebaseAuth _firebaseAuth = FirebaseAuth.instance;
  Future<String> signIn(String email, String password) async {
    FirebaseUser user = (await _firebaseAuth.signInWithEmailAndPassword(email: email, password: password)).user;
    return user.uid;
  }
  Future<String> createUser(String email, String password) async {
    FirebaseUser user = (await _firebaseAuth.createUserWithEmailAndPassword(email: email, password: password)).user;
    return user.uid;
  }
  Future<String> currentUser() async {
    FirebaseUser user = await _firebaseAuth.currentUser();
    return user != null ? user.uid : null;
  }
  Future<void> signOut() async {
    return _firebaseAuth.signOut();
  }
}
```

Listing 5. the file auth.dart

3. APPLICATION BUILD AND RELEASE

Unlike the codebase, the build process is specific for each platform. In fact, to build an android application, there should be a prepared release mainly by:

- Reviewing the app manifest;
- Reviewing the build configuration;
- Building the app for release.

The figure 2 shows the authentication screen.

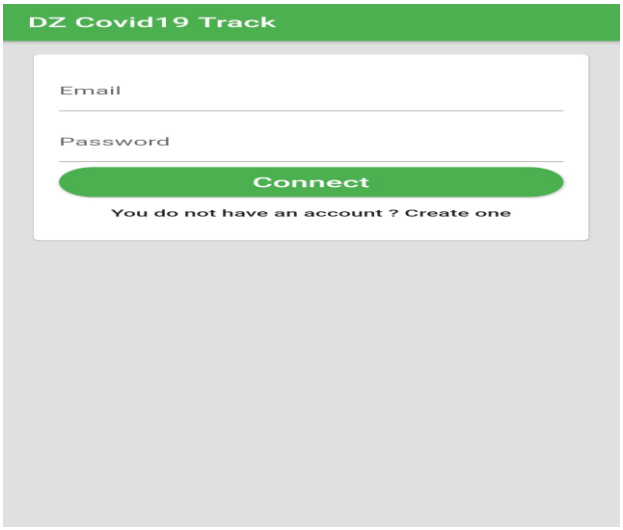

Figure 2. The authentication screen

The figure 3 shows the home screen.

Figure 3. The home screen

The figure 4 shows the diagnostic form screen.

Figure 4. The diagnostic form screen

The figure 5 shows the diagnostic result screen.

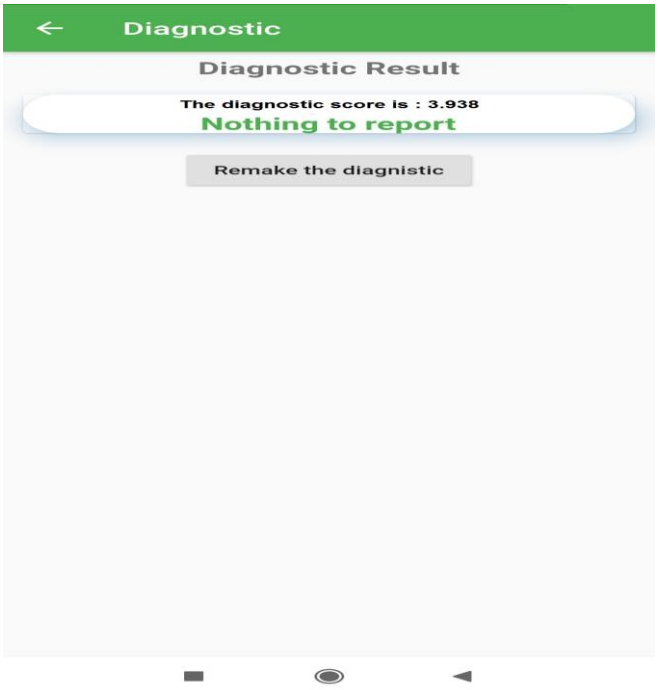

Figure 5. The diagnostic result screen.

4. CONCLUSION

In this paper, we have presented a cross-platform application for Covid 19 disease track. This application uses a list of questions with their weighting answers to calculate the score of responses given by a potential patient with Covid 19 disease symptoms. As the most stable platform in flutter and the most used by patients, Android is our main target.

As perspectives, we look to improve the application by adding other functionalities and testing them in the different platforms. Thus, we look to implement a machine learning model that enhances the algorithm used to calculate the score.

REFERENCES

[1] Hemanth, Jude D., et al. (2020) "An augmented reality-supported mobile application for diagnosis of heart diseases." The Journal of Supercomputing, Vol. 76, No. 2, pp1242-1267.
[2] Wood, Christopher S., et al. (2019) "Taking connected mobile-health diagnostics of infectious diseases to the field." Nature 566.7745 *: 467-474.
[3] Turbé, Valérian, et al. "Towards an ultra-rapid smartphone-connected test for infectious diseases." Scientific reports 7.1 (2017): 1-11.
[4] Haque, Farhana, et al. "Evaluation of a smartphone decision-support tool for diarrheal disease management in a resource-limited setting." PLoS neglected tropical diseases 11.1 (2017): e0005290.
[5] Chunara, Rumi, Clark C. Freifeld, and John S. Brownstein. "New technologies for reporting real-time emergent infections." Parasitology 139.14 (2012): 1843.
[6] Bissonnette, Luc, and Michel G. Bergeron. "Portable devices and mobile instruments for infectious diseases point-of-care testing." Expert Review of Molecular Diagnostics 17.5 (2017): 471-494.
[7] Lopez-Barbosa, Natalia, Jorge D. Gamarra, and Johann F. Osma. "The future point-of-care detection of disease and its data capture and handling." Analytical and bioanalytical chemistry 408.11 (2016): 2827-2837.
[8] Sharma, Shikha, Aoife Crawley, and Richard O'Kennedy. "Strategies for overcoming challenges for decentralised diagnostics in resource-limited and catastrophe settings." Expert review of molecular diagnostics 17.2 (2017): 109-118.
[9] Mao, Kang, Hua Zhang, and Zhugen Yang. "An integrated biosensor system with mobile health and wastewater-based epidemiology (iBMW) for COVID-19 pandemic." Biosensors and Bioelectronics 169 (2020): 112617.
[10] Xu, Lizhou, et al. "Facile biosensors for rapid detection of COVID-19." Biosensors and Bioelectronics (2020): 112673.
[11] Kumar, Aishwarya, Puneet Kumar Gupta, and Ankita Srivastava. "A review of modern technologies for tackling COVID-19 pandemic." Diabetes & Metabolic Syndrome: Clinical Research & Reviews (2020).
[12] Whitelaw, Sera, et al. "Applications of digital technology in COVID-19 pandemic planning and response." The Lancet Digital Health (2020).
[13] Chauhan, Nidhi, et al. "New and developing diagnostic platforms for COVID-19: A systematic review." Expert review of molecular diagnostics 20.9 (2020): 971-983.
[14] Udugama, Buddhisha, et al. "Diagnosing COVID-19: the disease and tools for detection." ACS nano 14.4 (2020): 3822-3835.
[15] Mainkar, Prajyot, and Salvatore Giordano. Google Flutter Mobile Development Quick Start Guide: Get Up and Running with IOS and Android Mobile App Development. Packt Publishing Ltd, 2019.