A Study of UI Construction in Android and Flutter: Comparative & Analysis

Donglan Zou[1,2,a,*], Mohamad Yusof Darus[1,b]

[1]College of Computing, Informatics and Mathematics, UiTM, Malaysia

[2]School of Mathematics and Computer Science, Xinyu University, Xinyu, Jiangxi, China

[a]29877624@qq.com, [b]yusof@fskm.uitm.edu.my

*Corresponding author

Keywords: UI Construction, Android, Flutter, Mobile Application, Performance Comparisons

Abstract: Android is a mobile operating system based on a modified version of the Linux kernel and other open-source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. At present, there are many technologies and frameworks that can be used for application development, especially mobile application development. Flutter is a popular user interface framework for mobile app development from Google. It has gained momentum over the past years. With the rapid growth of smartphones and tablets, learning mobile programming has become a crucial skill for students. Mobile app development involves writing a separate codebase for each platform. However, with the advent of cross-platform mobile programming, a paradigm shift has occurred in the mobile development landscape. Learning cross-platform mobile programming concepts and understanding their UI construction can be challenging, especially for beginners. This paper presents the study and comparison of UI construction, explicitly focusing on Android and Flutter framework. It starts by overview the Architecture of the two application development framework, then brief describes the process of building UI using Android and Flutter mobile application development tools. Followed by, it contains a case study of the applications developed by these technologies. At last, the differences in UI construction between Android and Flutter are summarized, and performance comparisons are conducted based on experimental data. The result has contributed to helping beginners quickly understand and master these two development tools.

1. Introduction

The usage of mobile devices like cellular phones, smartphones, and tablets has witnessed a significant surge in popularity over time. Alongside this rise in popularity, the availability of applications designed for these devices has also increased. In 2022, over 3.5 million mobile apps were available on the Google Play Store and approximately 2.2 million apps on the Apple App Store. In educations, mobile application development is an important field to study. In response to this demand, a large number of educational institutions started giving students opportunities to take mobile programming classes. The Association for Computing Machinery (ACM) recommends including topics related to mobile computing across the computer science curriculum.

To improve mobile programming education with self-study of students, Y. W. Syaifudin have developed a web application system that aims to learn mobile programming using the native programming language on the Android platform named Android programming learning assistance system(APLAS)[1] . Traditionally, mobile app development involved writing separate codebase for each platform, such as iOS (Apple) and Android (Google). However, with the advent of cross-platform mobile programming, a paradigm shift has occurred in the mobile development landscape.

Cross-platform mobile programming refers to the development of mobile applications that can run on multiple operating systems and platform using a single codebase. It leverages frameworks and tools that allow developers to write a code once and deploy it across different platforms, to save time and effort [2] . Therefore, it is important for students to study cross-platform app development.

Recently, Flutter has gained popularity as a software development kit for creating cross-platform applications compatible with Android and iOS. As a result, numerous software developers have

Copyright © (2024) Francis Academic Press, UK DOI: 10.25236/meeit.2024.027

embraced Flutter as their preferred choice. Abdul Rahman Patta studied the grammar-concept understanding problem (GUP) in mobile programming using Flutter frameworks, serving as an introductory exploration of Flutter programming for novice students [3].

For mobile application developers, both Android and Flutter should be familiar. Android is an operating system for mobile phones and tablets developed by Google based on the Linux platform. Since its inception, it has received unprecedented attention and quickly become one of the most popular operating systems on mobile platforms. Flutter is also an open-source mobile application development framework developed by Google. Its main goal is to provide a unified, high-performance, and easy to learn cross platform UI toolkit, allowing developers to build iOS and Android applications simultaneously using a single code repository. Flutter adopts a unique "responsive" architecture design, combined with *Dart* language and custom rendering engine, to achieve efficient interface drawing and animation effects.

In order to enable beginners to quickly understand and master these two development tools, this paper starts from the user interface design of the two, analyzes and compares the differences in the UI construction process of the two, and also analyzes and compares the differences in running speed and memory consumption under the same function implementation.

2. Foundations and Related Works

This section presents a brief overview of Android architecture and Flutter architecture, a brief describe of their UI construct process as well as the main contributions in the field of mobile development.

2.1. Literature Review

Since the inception of mobile devices, there have been several attempts to generate new processes for learning development of apps. The strategy so far has been to combine all existing methodologies or parts of them, with agile processes receiving the most attention.

In [4], Syaifudin et al. presented an implementation of an automated *Dart* code verification system for assisting mobile application programming learning using Flutter. Their work focused on developing a learning support system that integrates an automated Dart code verification mechanism, which draws upon a software testing methodology commonly used in Android application development.

In [3], Abdul Rahman Patta et al. Presented an implementation of grammar-concept understanding problem for cross-platform mobile programming learning, explicitly focusing on the Flutter framework. The evaluation of the implementation involved 109 undergraduate students in Makassar State University, Indonesia, who were assigned 22 instances covering 71 carefully selected keywords and questions. The results show that their proposal effectively reveals students understanding levels. These findings contribute to identifying areas for improvement and developing targeted learning resources to enhance mobile programming education.

In [5], Kishore, Khare et al. presented the study and comparison of the two most famous cross-platform application development technologies. It started by discussing the basic functions of the application development methodologies. Followed by, it contained a comparison of the performance of the applications developed by these technologies. After that Stability of applications made by the flutter and React Native is checked on different parameters. There was an implementation of an application using Flutter and React Native which will be used for performance analysis between two applications running Android and Web platforms.

In [6], Boukhary and Colmenares et al. proposed a new Flutter architecture based on the Clean Architecture by Uncle Bob. The Flutter Clean Architecture proposed in their research is packaged and released through a Flutter package. The architecture was tested by developing a full application from scratch using the package and documenting the process. The Flutter Clean Architecture provided a solution to the state management problem as well as a potential overall choice for Flutter mobile application architecture.

In [7], Choudhari, Gawai et al. proposed an app which is cost-effective as well as time saving. The

highlights of the project are that this app will give the statistical data of electricity used while simultaneously uploading the meter readings in the app and will also monitor the power consumption, Daily usage will be known to the user and this app would also provide the value of electricity units. User would be able to know the electricity bill till any time of the month and in addition to this, a report in the form of statistics and numbers will be updated daily in a journal. The prime reason behind developing such an app is to create an overall awareness about amount of electricity usage and consumption.

In [8], Perinello and Gaggi et al. studied if the cross-platform development frameworks Flutter and React Native allow to achieve accessibility of user interface of mobile applications. Their analysis shows that both frameworks do not provide a complete accessibility by default (RQ1), but they were able to find solutions to make components and widgets accessible (RQ2). Moreover, their analysis shows that React Native offers a more concise approach which leads to a briefer and thus readable source code (RQ3).

In [9], Martinez, Ferre et al. with the aim of defining a mobile application development framework that considers the specific characteristics of developing mobile apps, carried out a systematic mapping study of the software development process for mobile applications, then administered a survey and completed a qualitative study with industry experts. These studies identify the main trends in the software process for mobile apps, and to uncover the main challenges for app development. Their have organized the findings in a framework that integrates the specific challenges of mobile development, which called Mobile Ilities, with software development activities that are linked through an agile process. Their proposal has served as a guide for novice developers throughout the process of creating a final product, combining the existing knowledge of developers about Scrum with the specific characteristics of mobile development, and providing mechanisms to link these characteristics with the elements of the development process.

In [10], AMMAR et al. proposed a new approach and its support system for the automatic generation of mobile user interfaces. The approach and the system are based on a set of standards and relevant technologies such as EMF, GMF, ATL, and Xpand.

In the above literature, the authors' research has provided some assistance for students to learn mobile application development, but none of them have specifically studied UI construction for Android and Flutter.

2.2. Overview of Android Architecture

The underlying of the Android system is built on top of the *Linux* system, which consists of four layers: operating system, middleware, user interface, and application software. From top to bottom, the architecture of Android consists of four layers: Application layer, Application Framework layer, System Runtime layer, and Linux Kernel layer, as shown in Fig. 1.

Android applications are composed of components that can call independent functional modules. Components can be classified into four core components, namely *Activity, Service, BroadcastReceiver*, and *ContentProvider.* These four major components need to be registered with corresponding tags in the project's *AndroidManifest.xml* file. In an Android application, it mainly consists of four independent components that can be called and coordinated to form a true Android application [11].The communication of these components is mainly assisted by *Intent. Intent* is responsible for describing the actions, data involved, and additional data involved in a single operation in an application. Android component diagram as shown in Fig. 2.

Android, based on the description of this *Intent*, is responsible for finding the corresponding component, passing the Intent to the calling component, and completing the component call. Therefore, Intent plays a role as a media intermediary here, providing relevant information on component calls to each other, achieving decoupling between the caller and the called.

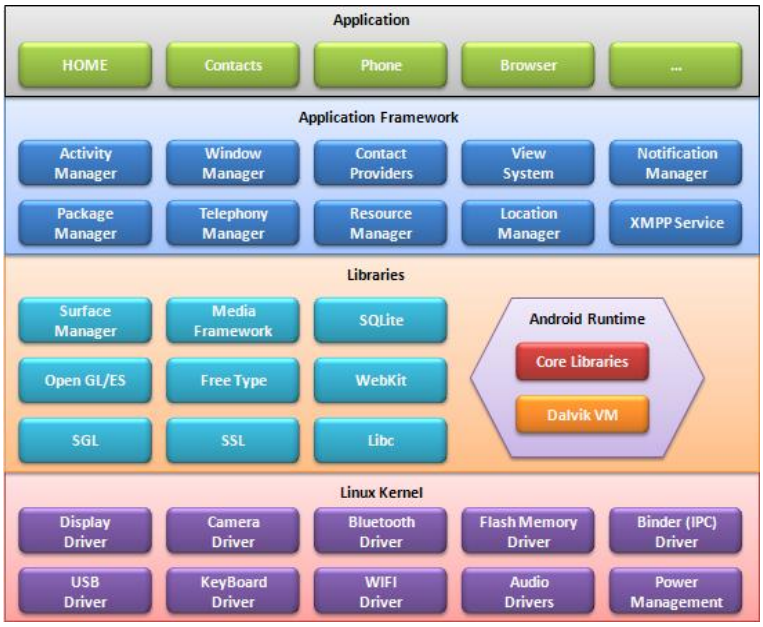

Fig. 1 Android Architecture Diagram.

Fig. 2 Android component diagram.

2.3. The File System Structure of Android Engineering

The directory structure of an Android application in Android Studio is shown in Fig. 3:

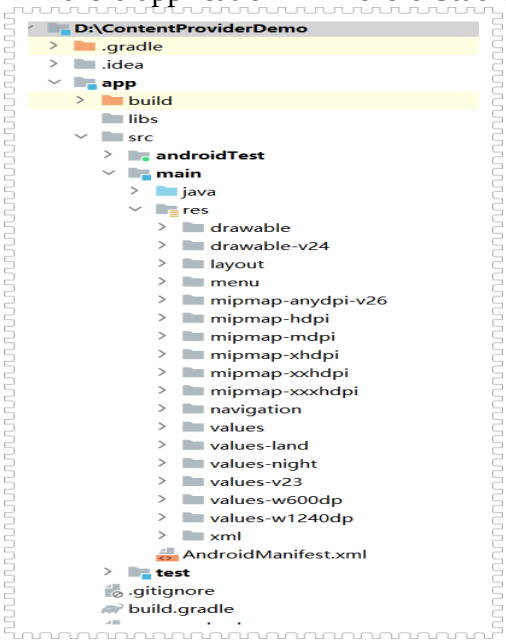

Fig. 3 Directory structure diagram of Android applications.

From Fig. 3, it can be obtained that Android application include Source Program Folder *src*,

Resource Folder *res*, Layout Folder *res/layout*, Value Folder *res/values*, Using the Extension Jar Package Folder *libs and* Engineering Project configuration list file *AndroidManifest.xml* these primary components. *AndroidManifest.xml* file contains relevant information that must be mastered before running the Android system, such as the application name, icon, package name of the application, component registration information, authorization, and the minimum Android version for running the device.

The entry point for running Android applications needs to be set to MAIN the *AndroidManifest.xml* file. For example, the code for defining *MainActivity.java* as the main activity of an application is as follows in Fig. 4:

```
<activity android:name=".MainActivity">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />

          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
```

Fig. 4 Code in AndroidManifest.xml file.

2.4. Hierarchy of Android UI Building

In Android, all UI elements are built through *View* and *ViewGroup*. For an Android application's user interface, *ViewGroup* serves as a container for the controls in the interface, which can include both regular *View* controls and *ViewGroup* controls[12].As shown in Fig. 5 and Fig. 6:

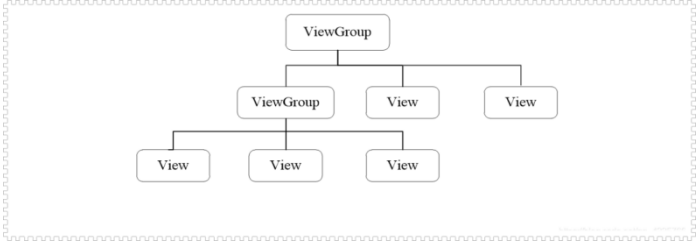

Fig. 5 Android View Hierarchy Chart.

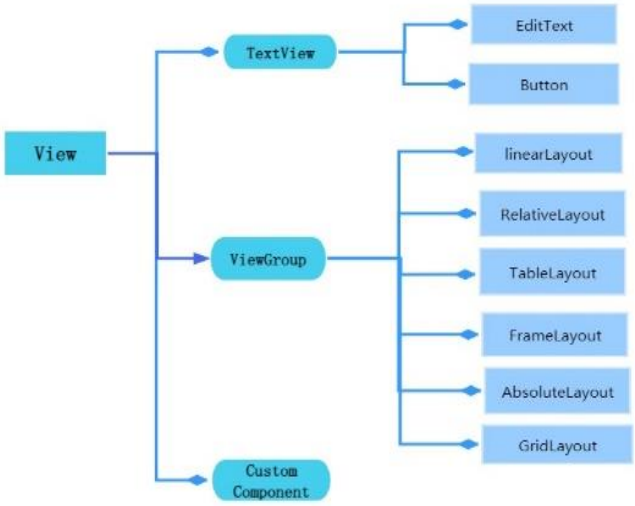

Fig. 6 Android UI Element Structure Diagram.

Interface layout writing method

Android has two interface layout methods. The first is to write the layout in an *xml* file, and the most commonly used is also this method. It can effectively isolate the code layout in the interface from *Java* code, making the structure of the program clearer, as shown in Fig. 7.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_centerHorizontal="true">
    <TextView
        android:layout_width="258dp"
        android:layout_height="120dp"
        android:layout_gravity="center"
        android:gravity="center"
        android:texeSize="80sp"
        android:text="Login"
        android:textColor="#ff0000"
        android:textSize="80sp">
    </TextView>
</LinearLayout>
```

Fig. 7 Writing layouts in XML files.

The second method is to write layouts in *Java* code, where all layout and control objects can be created using the "*new*" keyword. The created *View* control can be added to the *ViewGroup* layout to display the *View* control in the layout interface, as shown in Fig. 8.

```java
public class MainActivity1 extends Activity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout linearLayout=new LinearLayout( context: this);
        LinearLayout.LayoutParams params=new LinearLayout.LayoutParams( width: 258, height: 120);
        linearLayout.getOrientation();
        TextView textView=new TextView( context: this);
        textView.setText("Login");
        textView.setTextColor(Color.RED);
        textView.setTextSize(80);
        linearLayout.addView(textView,params);
        setContentView(linearLayout);
    }
}
```

Fig. 8 Writing layouts in Java files.

2.5. Flutter Architecture

Flutter is an open-source mobile application development framework developed by Google. Its main goal is to provide a unified, high-performance, and easy to learn cross platform UI toolkit, allowing developers to build iOS and Android applications simultaneously using a single code repository. Flutter adopts a unique "responsive" architecture design, combined with *Dart* language and custom rendering engine, to achieve efficient interface drawing and animation effects [13]. The system framework diagram is shown in Fig. 9.

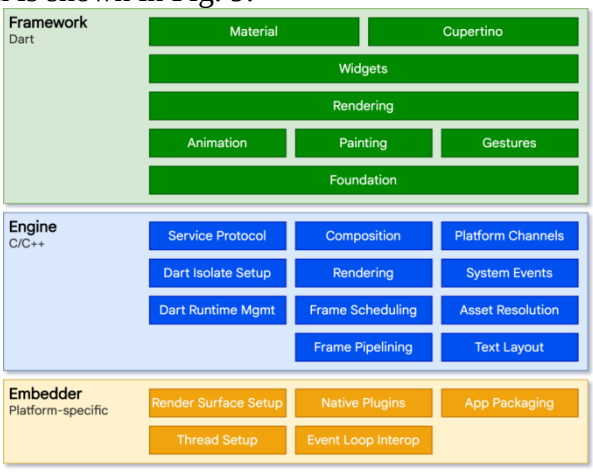

Fig. 9 Flutter system framework diagram.

From the above figure, we can see that the architecture of Flutter can be divided into three parts, from bottom to top, which are Embedded, Engine, and Framework. The Framework section is implemented in pure *Dart* language. The Foundation layer, along with the *Animation*, *Painting*, and

Gestures layers, provides animation, drawing, and gesture operations, which are specifically provided by Google for developers to call. The Rendering layer is responsible for building the UI tree, which means that when an Element on the UI tree changes, it recalculates the position and size of the changed part, updates the UI tree, and ultimately presents the updated interface to the user[14].

The application development of Flutter is mainly completed by the following 5 parts.

Dart language: Flutter is based on Google's own *Dart* programming language, which is a strongly typed, object-oriented, garbage collection language with concise syntax and easy to understand and get started with. *Dart* provides Flutter with rich API and library support for building UI components and handling business logic.

Widget system: The core concept of Flutter is "*Widget*", which is a reusable UI element or user interface component, such as buttons, text boxes, lists, etc. The *Widget* tree forms the view hierarchy of the entire application, with each *Widget* having its own state and build function (*build()*), responsible for generating and managing its own UI display. When the state of the *Widget* changes, Flutter will automatically rebuild the UI and update it by using the spreading algorithm to calculate the minimum change, thereby achieving efficient interface refresh [15].

Material Design & Cupertino Widgets: Flutter adopts two style systems, Material Design and Cupertino, corresponding to the design specifications of Android and iOS platforms, respectively. The collection of widgets under these two styles is called *Material and Cupertino Widget Libraries*, which encapsulate various UI components that match the characteristics of their respective platforms, allowing developers to quickly build application interfaces with platform characteristics [3].

Flutter Engine: Flutter's self-developed rendering engine, also known as Skia rendering engine, is the foundation for running Flutter applications. Skia is an open-source graphics library used to render high-performance 2D vector images and hardware accelerated images. Flutter Engine converts the *Dart* bytecode generated by the *Dart* compiler into a command sequence that Skia can understand, and drives Skia to draw UI elements on the GPU, achieving smooth animation effects and interactive experience [13].

Platform Channels: Flutter enables cross platform communication between Android and iOS through *Platform Channels*, allowing *Dart* code to interact with native layers (*Java or Objective-C*) and call native APIs to perform complex operations such as accessing device sensors, network requests, media playback, etc. Through this approach, Flutter ensures high performance even in scenarios with high performance requirements [15].

Hot Reloading & DevTools: Flutter has a built-in Hot Reloading feature, which allows for real-time preview and hot deployment of code changes without affecting the user experience. *DevTools* provides a powerful set of debugging tools, including viewing the *Widget* hierarchy, tracking state changes, analyzing memory leaks, and more, making it easy for developers to quickly locate and solve problems during the development process [16].

2.6. Hierarchy of Flutter UI Building

In flutter development, everything is a component, and all components are built on *Widgets*. The interface built by these *Widgets* constitutes the hierarchical structure diagram we see currently, as shown in Fig. 10.

The simplest Flutter application, just one *Widget*! As in the following example: pass a *Widget* to the runApp function, as shown in Fig. 11.

The runApp function takes the given *Widget* and makes it the root of the *widget tree*. In this example, the *widget tree* consists of two *Widgets: Center* (and its child widgets) and *Text.* The framework forces the root *widget* to cover the entire screen, which means the text "Hello, world" will be centered on the screen. The direction of text display needs to be specified in the *Text* instance. When using *MaterialApp*, the direction of the text will be set automatically.

When writing an application, developers usually create new *Widgets.* These *Widgets* are either *StatelessWidgets* or *StatefulWidgets.* The specific choice depends on whether your *widget* needs to manage some state. The main job of the *Widget* is to implement a build function to build itself. A W*idget* usually consists of a number of lower-level *Widgets*. The Flutter framework will build these

Widgets in sequence until the lowest-level sub-widget is built. These lowest-level *Widgets* are usually *RenderObject*, which calculates and describes the geometry of the *Widget* [17].

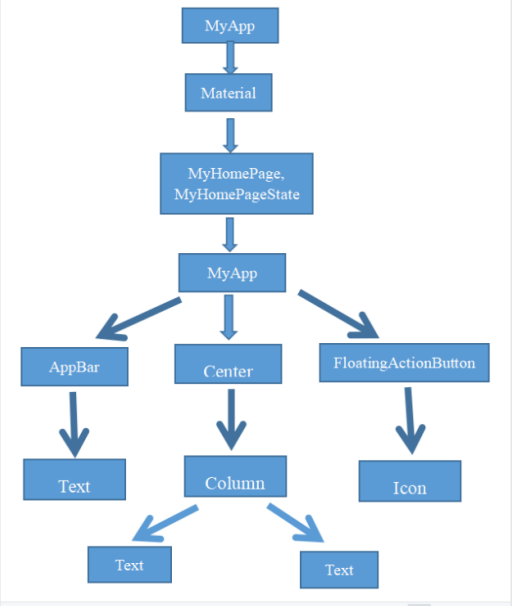

Fig. 10 Flutter program development hierarchy diagram.

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

void main() {
  runApp(
    new Center(
      child: new Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
}
```

Fig. 11 Simple flutter code example.

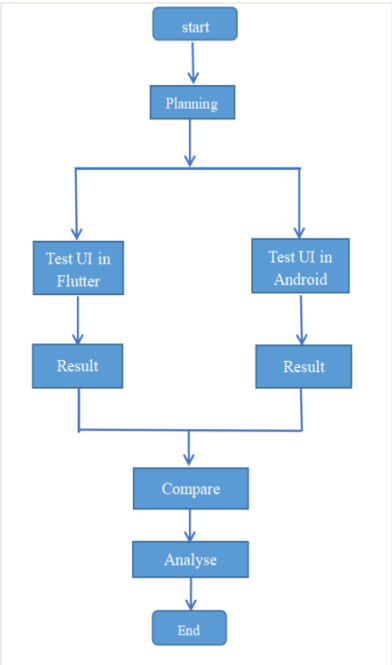

Fig. 12 Research Process Flowchart

3. Methodology

An experiment was carried out to find out the result of this research paper. This section presents the experiment and how it was prepared, its goal, and its process itself. Two applications are made for the Login function. one is using Android and another one is made using Flutter.

The research is executed in chronological phases as shown in Fig. 12 as Methodology Framework.

4. Testing & Result

4.1. UI Instance Generation Testing

This Login function contains two text input boxes and a "Login" button. One is for account input, and the other is for password input. The account includes two types: mobile phone number and email, which can simultaneously verify the input content.

Android UI code for implementing the Login function is shown in Fig. 13,and the running effect is shown in Fig. 14.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout   xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical"  android:layout_centerHorizontal="true">
    <TextView android:id="@+id/tv1" android:layout_width="258dp" android:layout_height="120dp" android:layout_gravity="center"
        android:gravity="center"  android:texeSize="80sp" android:text="Login"  android:textColor="#ff0000" />
    <EditText android:id="@+id/edtv1" android:layout_width="match_parent" android:layout_height="wrap_content" android:hint="Email Address" />
    <EditText  android:id="@+id/edtv2" android:layout_width="match_parent" android:layout_height="wrap_content" android:hint="Password" />
    <Button android:id="@+id/bt1" android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_gravity="center"  android:text="forget password"  android:textSize="40sp" />
    <Button  android:id="@+id/bt2" android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_gravity="center"  android:text="Sign in now"   android:textSize="40sp"/>
    <TextView  android:id="@+id/tv2"  android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:texeSize="40sp"  android:text="Login by other method" android:textColor="#ff0000" android:textSize="40sp"/>
    <LinearLayout android:layout_width="wrap_content" android:layout_height="wrap_content"  android:orientation="horizontal">
        <ImageView  android:layout_width="100dp"  android:layout_height="100dp" android:src="@drawable/facebook" />
        <ImageView  android:layout_width="100dp"  android:layout_height="100dp" android:src="@drawable/qq" />
        <ImageView  android:layout_width="100dp"  android:layout_height="100dp" android:src="@drawable/weichat" />
        <ImageView  android:layout_width="100dp"  android:layout_height="100dp" android:src="@drawable/google"  />
    </LinearLayout>
    <LinearLayout android:layout_width="match_parent" android:layout_height="100dp" android:gravity="center" android:orientation="horizontal">
        <TextView android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="No account?" android:textSize="30sp"/>
        <Button android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Register now" android:textSize="30sp"
            android:textColor="#FF00FF00"/>
     </LinearLayout>
  </LinearLayout>
```

Fig. 13 Android login interface code.

Fig. 14 Android login interface running renderings.

In flutter, the code to implement the login interface is shown in Fig. 15 and the running effect is shown in Fig. 16.

```
import 'package:flutter/material.dart';
class LoginPage extends StatefulWidget {  @override  _LoginPageState createState() => _LoginPageState();}
class _LoginPageState extends State<LoginPage> {  final _formKey = GlobalKey<FormState>();
  String _email='', _password='';  String value='';  bool _isObscure = true; Color _eyeColor=Colors.red;
  List _loginMethod = [{"title": "facebook","icon":Icons.facebook},{"title": "google","icon": Icons.search },
    {"title": "wechat","icon":Icons.wechat}];
    @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Form( key: _formKey,
        child: ListView( padding: EdgeInsets.symmetric(horizontal: 22.0),
          children: <Widget>[SizedBox(height: kToolbarHeight,),
            buildTitle(),buildTitleLine(),SizedBox(height: 70.0),
            buildEmailTextField(),SizedBox(height: 30.0),
            buildPasswordTextField(context),
            buildForgetPasswordText(context),SizedBox(height: 60.0),
            buildLoginButton(context),SizedBox(height: 30.0),
            buildOtherLoginText(),
            buildOtherMethod(context),
            buildRegisterText(context),
          ], ))); }  // <Widget>[], ListView, Form, Scaffold

Align buildRegisterText(BuildContext context) {
  return Align( alignment: Alignment.center,
    child: Padding(padding: EdgeInsets.only(top: 10.0),
      child: Row( mainAxisAlignment: MainAxisAlignment.center,children: <Widget>[Text('没有账号？'),
        GestureDetector(child: Text('点击注册',style: TextStyle(color: Colors.green),),
          onTap: () { print('去注册');Navigator.pushNamed(context,'/register');},),],),), ); }  // GestureDete

ButtonBar buildOtherMethod(BuildContext context) {
  return ButtonBar( alignment: MainAxisAlignment.center, children: _loginMethod
      .map((item) => Builder(builder: (context) {
      return IconButton(icon: Icon(item['icon'],color: Theme.of(context).iconTheme.color),
        onPressed: () {ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('第三方登录'),
          duration: const Duration(seconds: 2),action: SnackBarAction(label: '撤销',onPressed: () { },
          ),));}); },)).toList(),); }  // SnackBarAction, SnackBar, IconButton, Builder, ButtonBar

Align buildOtherLoginText() {return Align( alignment: Alignment.center,
  child: Text('其他账号登录',style: TextStyle(color: Colors.grey, fontSize: 14.0),));}  // Align

Align buildLoginButton(BuildContext context) {
  return Align(child: SizedBox(height: 45.0,width: 270.0,child:
  OutlinedButton(child: Text('Login',style: TextStyle(fontSize: 14.0, color: Colors.blue),),
    onPressed: () { if (_formKey.currentState!.validate()) { _formKey.currentState?.save();
      print('email:$_email , assword:$_password'); }},), ), ); }  // OutlinedButton, SizedBox, Align

Padding buildForgetPasswordText(BuildContext context) {
  return Padding( padding: const EdgeInsets.only(top: 8.0),
    child: Align( alignment: Alignment.centerRight,
      child: OutlinedButton(child: Text('忘记密码？',style: TextStyle(fontSize: 14.0, color: Colors.grey),),
        onPressed: () { Navigator.pop(context);},
      ), ), );  }  // OutlinedButton, Align, Padding

TextFormField buildPasswordTextField(BuildContext context) {
  return TextFormField( onSaved: (value) => _password = value!, obscureText: _isObscure,
    validator: (value) { if (value!.isEmpty) { return '请输入密码';  } },
    decoration: InputDecoration( labelText: 'Password',suffixIcon: IconButton(icon: Icon(Icons.remove_red_eye,color: _eyeColor,),
        onPressed: () {setState(() {_isObscure = !_isObscure;_eyeColor = (_isObscure?Colors.grey: Theme.of(context).iconTheme.color)!;});
        })), ); }  // IconButton, InputDecoration, TextFormField

TextFormField buildEmailTextField() {
  return TextFormField(decoration: InputDecoration(labelText: 'Email Address',),
    validator: ( value) { var emailReg = RegExp(
        r"[\w!#$%&'*+/=?^_`{|}~-]+(?:\.[\w!#$%&'*+/=?^_`{|}~-]+)*@(?:[\w](?:[\w-]*[\w])?\.)+[\w](?:[\w-]*[\w])?");  // RegExp
      if (!emailReg.hasMatch(value!)) {return '请输入正确的邮箱地址';} },
    onSaved: ( value) => _email = value!,  ); }  // TextFormField

Padding buildTitleLine() {
  return Padding(padding: EdgeInsets.only(left: 12.0, top: 4.0),
    child: Align( alignment: Alignment.bottomLeft,child: Container(color: Colors.black,width: 100.0,height: 2.0,),),);}  // Padding

Padding buildTitle() {
  return Padding( padding: EdgeInsets.all(8.0),child: Text('Login',style: TextStyle(fontSize: 42.0),),);}}
```

Fig. 15 Flutter login function code.

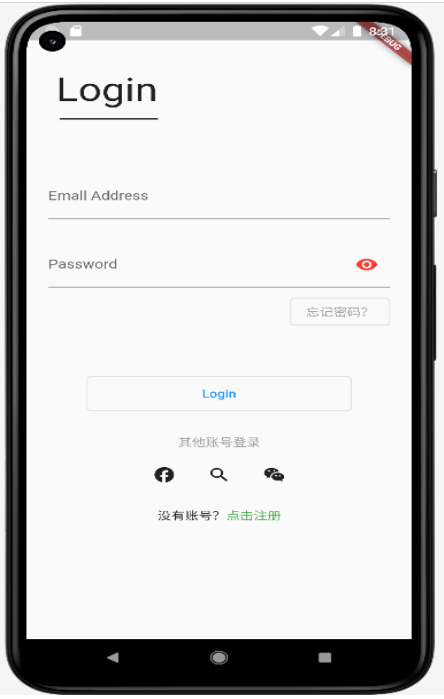

Fig. 16 Flutter login function code running renderings.

4.2. Experimental Result

The apps made using the various frameworks are compared on various parameters and the results are as follows [18].

1) Application Size

The applications run on the same system and their sizes were measured when they were loaded into the computer. Android app used the least amount of space in the system, while flutter apps used more space, as shown in Table 1.

Table 1 Application Size Comparison.

Application	App Size
Native Android	21.6MB
Flutter	193.3MB

2) Performance Comparison

The performance of the app is measured as per various parameters like CPU Usage, Memory Usage, Network Usage and Energy Usage. Table 2 is attached below indicating the same.

Table 2 Performance Comparison.

Application	CPU Usage	Memory Usage	Energy Usage
Native Android	100%	2.65KB	Very high
Flutter	25%	4.45KB	Very high

a) CPU Usage

The CPU usage is found out using the profiler of each platform. Android has the highest CPU usage, as shown in Table 2.

b) Memory Usage

The memory usage is found out using the profiler of the platform only. Table 2 shows that Android has the highest memory usage.

c) Energy Usage

Table 2 shows the energy usage comparison. It is found using profiler tool only. Both android and flutter

3) Code Reusability

Code Reusability is measured in the code by the number of times the same code is repeated throughout the whole code. In this case, as shown in Table 3 android is the best suited code as about 95 percent code is repeated, flutter have 90% reusability respectively.

Table 3 Various Parameter Comparison.

Technology Used	**App Size**	**Environment**	**Code Complexity**	**Package and it's installations**	**Code Reusability**
Native Android	Storage used is 21.6 MB.	Android Studio Version	Time complexity - O(n). Space complexity - n	Android 13.0	100 percent of code can be reused.
Flutter	Storage used is 139.3 MB.	Flutter 3.19.5	Time complexity - O(n). Space complexity - n	Flutter and dart plugins.	90 percent of code can be reused.

4) Code Complexity

It is determined using the function O(n) and due to the same logic implemented in all two codes, the complexity remains the same, that is O(n) and space complexity is n, where n is the number of moves taken. This is shown in Table 3.

5. Comparative & Analysis

5.1. UI interfaces are written differently

In Android, *xml* is usually used to write UI, and then it is parsed into a tree-structured UI tree through *LayoutInflater*. It also supports writing UI in code. The program is more readable. Convenient for program debugging and modification.

There is no *xml* layout file in Flutter, but layout through *Widget* tree. Flutter can only construct a UI tree through code, representing an interface. In Android, we can directly modify *Views* to update them. However, in Flutter, *Widgets* are immutable and cannot be updated directly. Instead, the *Widget*'s state is modified.

Widgets in Flutter are divided into two types: *stateful* and *stateless*.

- *StatelessWidget* stateless control. A UI that once created will not change on the fly.
- *StatefulWidget* with state control. UI that needs to be dynamically modified depending on external information after creation.

5.2. The components of the UI interface are different

In Android, the component element of the UI interface is *View*. All interface elements inherit the *View* class and are derived from *View*. The component elements of the Flutter UI interface are *Widgets*, and any interface element inherits *Widgets*.

What does *View* correspond to in Flutter?

In the Android framework, *View* is the basis for all content displayed on the screen, and all controls (*Button*, *TextView*, etc.) are a *View*. In Flutter, *Widget* can be roughly regarded as the equivalent of *View*, because it cannot be completely mapped to *View* in Android.

Widgets have no life cycle and are immutable once created. When a *Widget* needs to be changed, Flutter will rebuild a new *Widget* instance tree. *Widget* itself is not a *View*, it does not draw anything directly, it is just a description of the UI and its semantics.

How do they add or delete files in the layout?

In Android, subviews can be dynamically added and deleted through the *addChild()* or *removeChild()* method of the parent view.

In Flutter, the drawn content can be modified through the Boolean value and return value of the

parent *View*.

5.3. UI interface elements have different variability

View in Android is variable. When user interacts or data is updated, the *invalidate* method of *View* can be directly called to redraw to update the UI.

Widget itself in Flutter is immutable. So how does Flutter update the interface? Flutter updates the interface through the *StatefulWidget* control object.

The base class of statefulwidget is *StatefulWidget*, which is associated with a *State* object and saves the state information of the *Widget*, such as whether it is currently selected, etc. *State* information can also be saved in the *Widget*, and then *State* obtains the state information through the *Widget.* The *createState* function of the *statefulwidget* is responsible for creating its own associated *State* object. The *statefulwidget* entrusts its own construction to the *State* object. The build function of the *State* object is responsible for building the *Widget*. When the user interacts or the data changes, the *Widget* state changes. The *setState* method of *State* is called to notify it, and then the *State* is based on the current state information, rebuild the *Widget tree*.

5.4. Other differences

The difference between the UI construction of Android and Flutter is not only the components used in the above experiments, but also differences in *Custom Widgets*, dynamic addition of *Child Widgets*, *Canvas Drawing* and *Gesture Monitoring*, as shown in Table 4.

Table 4 Other differences between Android & Flutter.

Application	Custom Widget	Dynamically adding child widgets	Canvas drawing	Gesture monitoring
Native Android	*inherit View*	*addView()/removeView()*	inherit *View*, override *onDraw()*	inherit *View*, override *onTouchEvent()*
Flutter	inherit *StatelessWidget*/ *StatefulWidget*, Override *build()*	two sets of *Widget tree*	*CustomPaint*	*GestureDetector*

From the Table 4, implement *Custom View* by inheriting an existing *View* in Android, while inherit *StatelessWidget*/*StatefulWidget*, and override *build()* method in Flutter. Regarding the dynamic addition of *Child Widgets*, it can be added/removed at any time through *addView/removeView* method in Android. For Flutter the parent *Widget* can only prepare two sets of *Widget tree* to implement this function. In Android it can inherit *View* and override *onDraw* method to customize the drawing through *Canvas,* while Flutter use *CustomPaint* class. Regarding the *Gesture monitoring*, Android inherit *View*, override *onTouchEvent()* method, while Flutter use *GestureDetector* class.

6. Conclusion

This paper elaborates on the process of building UI using two mobile application development platforms, Android and Flutter, and compares and analyzes the differences between them.Their differences are:

- ◆ UI interfaces are written differently.

In Android it usually uses *xml* to construct UI, and it also supports building UI in *java* code. The program is more readable and convenient for program debugging and modification. In Flutter it can only construct a UI by *Widget.*

- ◆ The components of the UI interface are different.

All interface elements inherit the *View* class and are derived from *View*. The component elements of the Flutter UI interface are *Widgets*, and any interface element inherits *Widgets*.

- ◆ UI interface elements have different variability

View in Android is variable. When user interacts or data is updated, the *invalidate* method of *View* can be directly called to redraw to update the UI. *Widget* itself in Flutter is immutable. Flutter updates

the interface through the *StatefulWidget* control object.

◆ Different ways to implement *Custom Widget*

Implement *Custom View* by inheriting an existing *View* in Android, while inherit *StatelessWidget*/*StatefulWidget*, and override *build()* method in Flutter.

◆ Different in dynamically adding *Child Widgets*

It can be added/removed at any time through *addView/removeView* method in Android. For Flutter the parent *Widget* can only prepare two sets of *Widget tree* to implement this function.

◆ Different ways to implement *Canvas Drawing*

In Android it can inherit *View* and override *onDraw* method to customize the drawing through *Canvas*, while Flutter use *CustomPaint* class.

◆ Different ways to implement *Gesture Monitoring*

Android inherit *View*, override *onTouchEvent()* method, while Flutter use *GestureDetector* class to achieve the *Gesture Monitoring* function.

This paper also discusses the test results obtained after implementing the application. the obtained indicated that Android has the smallest application size, while flutter has the largest application size. Comparing the code complexity of the development code of each of the applications, it obtained that the highest code reusability percentage is in Native Android, followed by Flutter.

This study's outcomes have contributed to helping beginners quickly understand and master these two development tools. However, this paper only discussed the difference on UI construction between Android and Flutter. There are many differences between flutter and Android. In future work, the researcher will focus on studying the differences between Android and Flutter in loading images, animations, and other aspects.

Acknowledgements

(1) National Natural Science Foundation of China (No. 62062063)

(2) The Science and Technology Research Project of Jiangxi Provincial Department of Education, China (No. GJJ202310);

(3) The Jiangxi Provincial Natural Science Foundation, China (No.20224BAB202022)

References

[1] Y. W. Syaifudin, N.F., M. Kuribayashi, and W.-C. Kao, A proposal of Android programming learning assistant system with implementation of basic application learning. Int. J. Web Inform. Sys., 2019. 6.

[2] Mahendra, M. and B. Anggorojati, Evaluating the performance of Android based Cross-Platform App Development Frameworks, in Proceedings of the 6th International Conference on Communication and Information Processing. 2021, Association for Computing Machinery: Tokyo, Japan. p. 32–37.

[3] Abdul Rahman Patta, N.F., Xiqin Lu, Yan Watequlis Syaifudin, A Study of Grammar-concept Understanding Problem for Flutter Cross-platform Mobile Programming Learning. 2023 6th International Conference on Vocational Education and Electrical Engineering (ICVEE), 2023. 2023 6th.

[4] Syaifudin, Y. W., et al. (2022). An Implementation of Automatic Dart Code Verification for Mobile Application Programming Learning Assistance System Using Flutter. 2022 International Conference on Electrical and Information Technology (IEIT): 322-326.

[5] Kishore, K., et al. (2022). "Performance and stability Comparison of React and Flutter: Cross-platform Application Development." 2022 International Conference on Cyber Resilience (ICCR): 1-4.

[6] Boukhary, S. and E. Colmenares (2019). A Clean Approach to Flutter Development through the Flutter Clean Architecture Package. 2019 International Conference on Computational Science and

Computational Intelligence (CSCI).

[7] Choudhari, A., et al. (2022). A Mobile App for Smart Electricity Usage Monitoring. 2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS): 1667-1673.

[8] Perinello, L. and O. Gaggi (2024). Accessibility of Mobile User Interfaces using Flutter and React Native. 2024 IEEE 21st Consumer Communications & Networking Conference (CCNC): 1-6.

[9] Martinez, D., et al. (2020). "An Agile-Based Integrated Framework for Mobile Application Development Considering Ilities." IEEE Access 8: 72461-72470.

[10] AMMAR, L. B. (2021). "An Automated Model-Based Approach for Developing Mobile User Interfaces." IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 9.

[11] Mayrhofer, R., et al. (2021). "The Android Platform Security Model." ACM Transactions on Privacy and Security 24(3): 1-35.

[12] Jackson,W. (2014). "Android Apps for Absolute Beginners." New York, NY:Apress.

[13] Durai, S., et al. (2022). Cloud Computing based Multipurpose E-Service Application using Flutter. 2022 6th International Conference on Computing Methodologies and Communication (ICCMC): 1122-1126.

[14] Nagaraj, K., et al. (2022). Application Development for a Project using Flutter. 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC): 947-951.

[15] Syaifudin, Y. W., Hatjrianto, A. S., Funabiki, N., Liliana, D. Y., Kaswar, A. B., & Nurhasan, U. (2022). An Implementation of Automatic Dart Code Verification for Mobile Application Programming Learning Assistance System Using Flutter. Paper presented at the 2022 International Conference on Electrical and Information Technology (IEIT).

[16] Aakanksha Tashildar, N. S., Rushabh Gala, Trishul Giri, Pranali Chavhan. (2020). Application Development for a Project using Flutter. International Research Journal of Modernization in Engineering Technology and Science, 02(08).

[17] Syaifudin, Y. W., et al. (2024). Implementation of Self-Learning Topic for Developing Interactive Mobile Application in Flutter Programming Learning Assistance System. 2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS): 1103-1107.

[18] Suri, B., et al. (2022). Cross-Platform Empirical Analysis of Mobile Application Development frameworks: Kotlin, React Native and Flutter. Proceedings of the 4th International Conference on Information Management & Machine Intelligence: 1-6.