Name: Kanish Jadhwani Roll No.: 22 Class: D15B

EXPERIMENT NO.3

Experiment No 3: **Exploring Flutter Widgets**	
ROLL NO	22
NAME	Kanish Jadhwani
CLASS	**D15-B**
SUBJECT	**MAD & PWA Lab**
LO-MAPPED	

Aim : To explore Flutter Widgets like image,icon and to use custom fonts.

Theory :

Images:

Flutter is an open-source, cross-platform UI development kit developed by Google. It is gaining popularity these days, as the app made in flutter can run on various devices regardless of their platform. It is majorly used to develop applications for

Android and iOS, as a single app made in flutter can work efficiently on both platforms.

An asset is a file, which is bundled and deployed with the app and is accessible at runtime. The asset can include static data, configuration files, icons, and images. Flutter supports many image formats, such as JPEG, WebP, PNG, GIF, animated WebP/GIF, BMP, and WBMP.

Displaying images is the fundamental concept of most of the mobile apps. Flutter has an Image widget that allows displaying different types of images in the mobile application.

Icons:

An icon is a graphic image representing an application or any specific entity containing meaning for the user. It can be selectable and unselectable. For example, the company's logo is non-selectable. Sometimes it also contains a hyperlink to go to another page. It also acts as a sign in place of a detailed explanation of the actual entity.

Flutter provides an Icon Widget to create icons in our applications. We can create icons in Flutter, either using inbuilt icons or with the custom icons. Flutter provides the list of all icons in the Icons class. In this article, we are going to learn how to use Flutter icons in the application.

Icon Widget Properties:

Flutter icons widget has different properties for customizing the icons. These properties are explained below:

Fonts :

Download Font File: A Font file has all the required details regarding a font family, so once imported the font style can be used throughout the app. Google Fonts website has a wide variety of font families that can be downloaded and used in an app. The steps for downloading the file are as follows:

Step 1: Open Google Fonts and search for a font family in the search bar (here

“Pacifico”).

Step 2: Select the “Pacifico” font file.

Step 3: To download, click the “Download Family” button.

Import Font Files: To use the downloaded font family, we need to import the font file into the root of the Flutter app. The steps for importing the font file in Android Studio are as follows:

Step 1: Click the “Project” button in the top left corner of Android Studio.

Step 2: Right-click on the project name, here “gfg_custom_fonts” and select New + Directory.

Step 3: Name the directory as “fonts“.

Step 4: Open file manager and go to downloads. Extract and open the “Pacifico” zip file.

Step 5: Move the “Pacifico Regular” file into this directory. After moving, the font directory contains a file named “Pacifico-Regular.ttf”.

Declare Font: Now after importing the font file, it’s a necessity to tell Flutter where to fetch the font file from. So, it’s a need to declare the font in a specific file named “pubspec.yaml” file. Indentation plays a major role in this file, where double-space is used to indent the code. The steps to declare the font file is as follows:

Step 1: Click the “Project” button and click on the project name, here “gfg_custom_fonts”.

Step 2: In the list, there is a file called “pubspec.yaml” file. Click this file.

Step 3: Paste the below code in the pubspec.yaml file. Be aware of the indentations.

Step 4: Press the “Pub get” button in the top right corner of the file.

Add Font Family: Now this font family can be used in styling the text widget in the app wherever needed. Not only one, but multiple families can be downloaded in the above-mentioned steps.

The syntax for it is as follows:

Text('text', style: TextStyle(

fontFamily:

'family_name',)

Code:

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

final themeNotifierProvider = StateNotifierProvider<ThemeNotifier,
ThemeData>((ref) {
  return ThemeNotifier();
});

class Pallete {
  // Colors
  static const blackColor = Color.fromRGBO(1, 1, 1, 1); // primary color
  static const greyColor = Color.fromRGBO(26, 39, 45, 1); // secondary color
  static const drawerColor = Color.fromRGBO(18, 18, 18, 1);
  static const whiteColor = Colors.white;
  static var redColor = Colors.red.shade500;
  static var blueColor = Colors.blue.shade300;

  // Themes
  static var darkModeAppTheme = ThemeData.dark().copyWith(
    scaffoldBackgroundColor: blackColor,
    cardColor: greyColor,
    appBarTheme: const AppBarTheme(
      backgroundColor: drawerColor,
      iconTheme: IconThemeData(
        color: whiteColor,
      ),
    ),
    drawerTheme: const DrawerThemeData(
      backgroundColor: drawerColor,
```

```
    ),
    primaryColor: redColor,
    backgroundColor: drawerColor, // will be used as alternative background
color
  );

  static var lightModeAppTheme = ThemeData.light().copyWith(
    scaffoldBackgroundColor: whiteColor,
    cardColor: greyColor,
    appBarTheme: const AppBarTheme(
      backgroundColor: whiteColor,
      elevation: 0,
      iconTheme: IconThemeData(
        color: blackColor,
      ),
    ),
    drawerTheme: const DrawerThemeData(
      backgroundColor: whiteColor,
    ),
    primaryColor: redColor,
    backgroundColor: whiteColor,
  );
}

class ThemeNotifier extends StateNotifier<ThemeData> {
  ThemeMode _mode;
  ThemeNotifier({ThemeMode mode = ThemeMode.dark})
      : _mode = mode,
        super(
        Pallete.darkModeAppTheme,
      ) {
    getTheme();
  }

  ThemeMode get mode => _mode;

  void getTheme() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    final theme = prefs.getString('theme');

    if (theme == 'light') {
      _mode = ThemeMode.light;
      state = Pallete.lightModeAppTheme;
    } else {
      _mode = ThemeMode.dark;
```

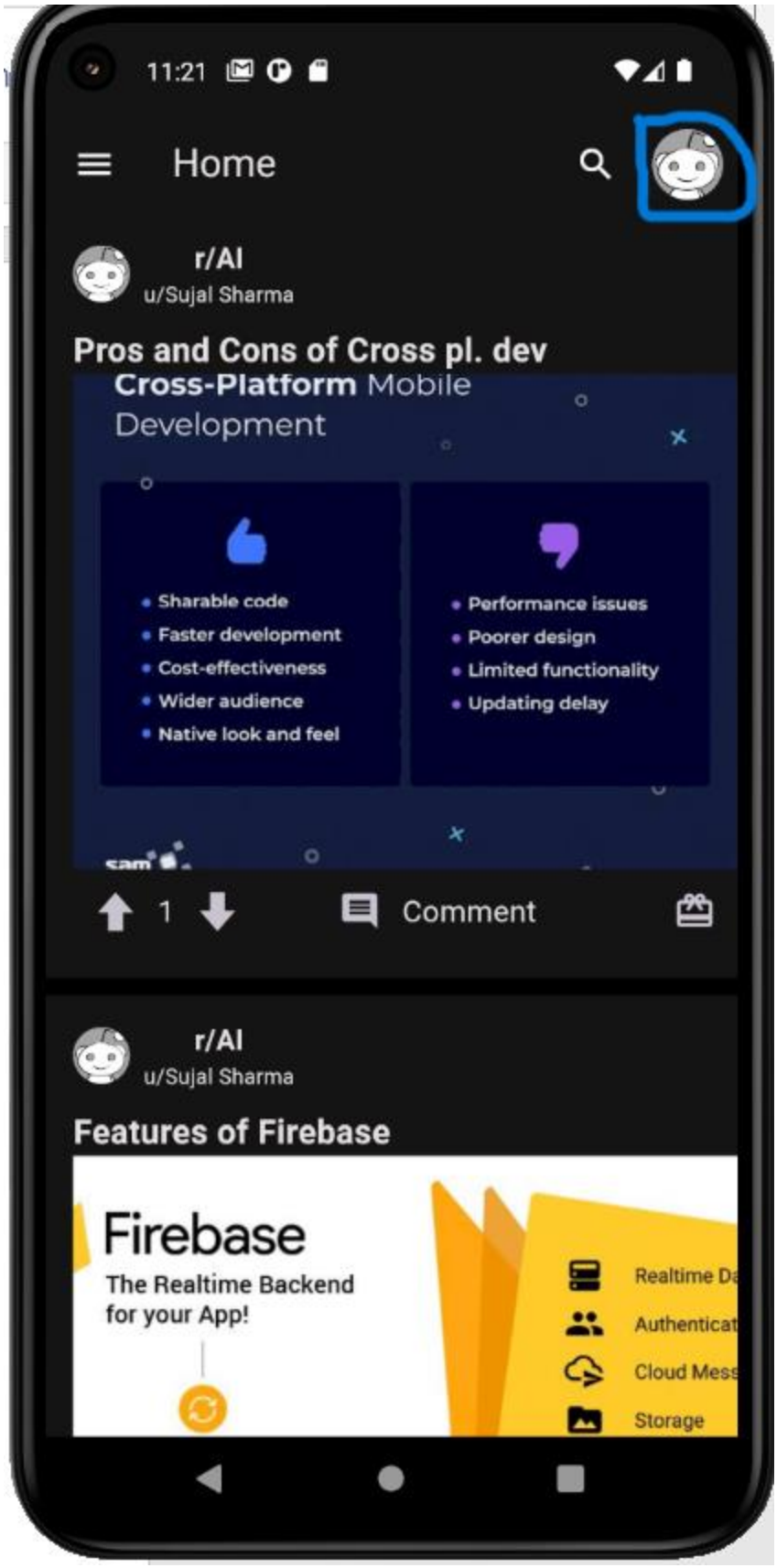
```
      state = Pallete.darkModeAppTheme;
    }
  }

  void toggleTheme() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();

    if (_mode == ThemeMode.dark) {
      _mode = ThemeMode.light;
      state = Pallete.lightModeAppTheme;
      prefs.setString('theme', 'light');
    } else {
      _mode = ThemeMode.dark;
      state = Pallete.darkModeAppTheme;
      prefs.setString('theme', 'dark');
    }
  }
}
```

Output:

Name: Kanish Jadhwani Roll No.: 22 Class: D15B



Name: Kanish Jadhwani Roll No.: 22 Class: D15B

Conclusion:

We understood and implemented the images ,icons and fonts in flutter which displayed the versatile features of flutter and enhanced the user Interface.While the experiment was successful in achieving its objectives, it's essential to consider the potential challenges. Care should be taken in optimizing image sizes, to maintain optimal loading times. Additionally, thorough testing across different devices and screen sizes is crucial to ensuring a responsive and consistent user experience.