Name - Prathamesh Shetty
Class - D15B Roll No: 53

MAD Assignment 1

A1 04

Q.1

(a) Explain the key features and advantages of using flutter for mobile app. development.

Ans ① Single Codebase for multiple platforms -
Write one codebase for both android and ios, reducing development effort and maintenance.

② Hot Reload -
Instantly see changes in the app without restarting making development faster and more interactive.

③ Fast Performance - ~~Used~~ Uses the dart language and a compiled approach for smooth and high performance apps.

④ Open source and strong community support -
Backed by *google and a large developer community, ensuring continuous improvements and resources.

Advantages:

① Faster Development Time: Hot reload and single ~~code~~ codebase reduce development time significantly.

② Cost Effective: Since the code runs on both android and ios, businesses save on development and maintenance.

③ Reduced performance issues: The app runs natively without relying on intermediate bridges like in react native reducing lag.

b. Discuss how the flutter framework differ from th traditional approaches and why it has gained popularity in the dev community?

Ans ① Single Codebase vs. Seperate Codebase

- Traditional approach: Developers need to write seperate code for Android and iOS
- Flutter: Uses single dart codebase for both platforms reducing dev time.

② Rendering Engine vs Native UI components

Traditional Approach: Relies on platform native UI components which can leads to inconsistencies and performance issues.

Flutter: Uses the skia rendering engine to draw everything from scratch ensuring a consistent UI across devices.

Why flutter has gained popularity?

① Faster Developement with Hot Reload - Instantly see UI changes after code change.

② Cross Platform Efficiency - Single dart code base for Android and IOS.

③ Consistent UI - Flutter do not rely on native UI components.

④ Improved Performance - AOT compilation and direct access to GPU ensure smooth animations.

⑤ Easy integration with Backend Technologies: Works well with Firebase, RESTful API's, GraphQL and other backend.

Q.2.

(a) Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces.

-> Widget Tree in Flutter.

In Flutter, the widget tree is the fundamental structure that represents the UI of an application. It is the hierarchial arrangement of widgets where each widget defines the part of a the user interface. Flutters ~~UI~~ UI is entirely built using widgets which can be stateless or stateful. The widget tree determines how the UI is rendered and updated when changes occur.

Widget Composition in Flutter

Widget composition refers to building complex UI by combining smaller, reusable widgets, instead of creating large, monolithic UI components. Flutter encourages breaking the UI into smaller manageble widgets.

```
Example: class ProfileCard extends StatelessWidget {
           final String name;
           final String imageUrl;
           ProfileCard ({required this.name, required this.imageUrl});

           @Override
           Widget build(BuildContext context) {
             return Card(
               child: Column(
                 children: [
```

```
Image.network (ImageUrl);
SizedBox(height:10),
Text(name, style: TextStyle(fontSize: 20, fontWeight:10),
]
);
}
```

Benefits of widget composition:

① Reusability: Widgets can be reused.

② Maintainability: Breaking UI into smaller components makes it easier to maintain.

③ Performance: Flutter efficiently rebuilds only necessary part of widget tree.

b) Provide examples of commonly used widgets and their roles in creating a widget tree.

① Structural Widgets:

These widgets act as the foundation of building the UI

- Material App: The root widget of a flutter app that provides essential configurations.
- Scaffold: Provides a basic layout structure, including an app bar, body, floating action button, etc.
- Container: A versatile widget used for styling, padding, margins and background customization.

Ex.
```
MaterialApp(
    home: Scaffold(
        appBar: AppBar(title: Text("Flutter Widget Tree")),
        body: Container(
            padding: EdgeInsets.all(16.0),
            child: Text("Hello, Flutter!"),
        ), // Container
    ), // Scaffold
), // MaterialApp
```

2. ② Input & Interaction Widgets

TextField - Accepts text input from users

Elevation buttons - A button with elevation.

Gesture Detector - Detects gets gestures like taps, swipes and longPress.

Ex. Column(

children: [

TextField (decoration: InputDecoration (labelText: "Enter Name"),

ElevatedButton(

onPressed: () {

print ("Button Pressed");

},

child: Text ("Submit"),

),

],

),

Q.3.

Q) Discuss the importance of state management in flutter applications

-> In flutter, State refers to data that ~~can~~ can change during the lifetime of an application. This includes:

- User input
- UI changes
- Network changes
- Animation States

There are two types of States:-

① Ephemeral States - Small, UI specific state that do not affect the whole app.

② App Wide States - Data shared across multiple widgets

Importance of state Management -

- Efficient UI updates - Flutters UI is rebuilt whenever state changes.
- Code Maintainability and Scalability :- Managing State properly makes the code modular.
- Data Consistency and Synchronization :- Proper State management ensures that data remains consistent across different screens.

b. Compare and Contrast the different state management approaches available in flutter, such as set State, Provider and RiverPod. Provide scenarios where each approach is suitable.

Ans 1. set State - Local State

Pros - Simple built in case to use

Cons - Not scalable, causes unnecessary re-renders.

Best Use cases - Small UI updates (eg. toggle Switch, counter).

Provider - App wide State

Pros - lightweight, recommended by flutter, efficient

Cons - Boilerplate code for nested Providers.

Best Use cases - Medium Scale apps (eg. authentication, themes, API data)

RiverPod - App-wide state (more scalable than Provider)

Pros - Eliminates providers limitations, improves Performance.

Cons - Requires learning new concepts

best use case - Large apps needing global state (eg. Shopping Cart, user sessions).

Scenarios for each approach

- Use state while managing simple UI elements within a single widget like toggling dark mode in a setting screen.
- Use provider when sharing state across multiple widgets such as managing user authentication
- Use riverpod when creating a complex scalable app.

Q.4.

(a) Explain the process of integrating firebase with a flutter application. Discuss the benefits of using Firebase.

-> Firebase provides a powerful backend solution for further application offering services like authentication, real time databases, cloud functions, storage and more.

Steps to integrate firebase with flutter:

(1) Create a firebase project
- Go to firebase console
- Click 'Add Project' and enter name
- Configure google analytics

(2) Register the flutter app with firebase
- Click "Add App" and select Android or ios.
- For android, enter the android package name and download the google services

(3) Install firebase dependencies.
- Add following in pubspec.yaml file
firebase core, firebase auth, cloud firestore
- Run flutter pub get

④ Configure Firebase for Android or iOs.
- Opens android/ build.gradle and ensure 'com.google.gms:google-services:4.3.10'

⑤ Initialize firebase in flutter

Benefits of using firebase

Firebase is a Backend-as-a-Servie (Baas) that simplifies backened development for further apps. Here are some Key benefits.

① Easy to setup and scale
② Provides user authentication
③ Cloud storage
④ Push notifications
⑤ NoSQL database

(b) Highlight the firebase services commonly used in flutter development and provide a brief overview of how data synchronisation is achieved.

→ Firebase provides a suite of backend services that simplify flutter app development

① Firebase Authentication
Enables secure authentication using email/password, phone, and third party providers like google, facebook and Apple.

② Cloud Firestore

Stores and syncs data in real time across devices. Supports structured data, queries and offline access.

eg. Firebase.firestore().collection('users').add({
'name': 'John Doe';
'email': 'johndoe@example.com',
});

③ Realtime Database

A realtime json based database that automatically updates data across devices

ex- Database reference ref = FirebaseDatabase.instance.ref("message");
ref.set({"text": "Hello, Firebase!"});

④ Firebase cloud messaging

Enables push notifications and ~~auth~~ messaging between users.

⑤ Firebase Analytics

Tracks user interactions and app performance

⑥ Firebase Hosting

Deploys and serves web applications securely with automatic SSL.

Data Synchronization in Firebase

Firebase ensures real time data synchronization across multiple devices.

① Cloud Firestore Sync mechanism

Uses real time listeners to update UI instantly when data changes.

Ex- Firebase.Firestore.instance.collection("users").snapshots().listen(snapshot) {

```
for (var doc in snapshot.docs) {
    print(doc['name']);
}
});
```

② Real time Database Sync Mechanism.

Uses persistent web socket connections for live updates.

③ Offline data sync

Firebase caches data locally and syncs changes when the device is online.

④ Cloud functions for automated updates

Automates backend logic to trigger updates when data changes.