Name - Tarun Mishra Div - D15B
Roll No - 33 MPL Assignment 1

Q(1)
Ans(a) Key Features and Advantages of Flutter for Mobile App Development

- Single Codebase for Multiple Platforms: One code to run Web, iOS, Android, etc.
- Hot Reload: Instantly see changes in app without losing state, speeding up development and debugging
- Rich UI with Customizable Widgets: Provides a wide range of pre-design widgets and allows full customizatio to build complex, responsive UIs
- High Performance: Built on Dart, Flutter compiles to native ARM code, eliminat the need for a Javascript bridge.
- Access to Native Features: Flutter offer plugins and platform channels to acce native device features like GPS, camera, sensors, etc.
- Google Support: Backed by Google, Flutter receives regular updates, & stability improvements and stable long-term support.

Teacher's Sign.: _____

b.) Rendering Approach: Traditional frame
rely on native components, but Flut
draws everything itself using Ski
This eliminates dependency on OEM wid
and prevents UI inconsistencies.

- Close - Platform without sacrificing
Performance: Unlike frameworks li
React Native, which use a bridge to
communication with native compon
Flutter's direct compilation to nativ
code avoids performance bottlenec

Declarative UI: Flutter uses a reacti
declarative UI approach where UI
automatically updates when app state
changes - similar to modern front-e
frameworks.

Reduced Maintenance Effort: A single
codebase means fewer bugs and eas
updates, as developers don't need to
maintain seperate code for iOS and
Android.

- Faster Development Cycles: Hot reload
and a vast library of ready to
use components drastically cut
down development and testing time

Teacher's Sign.: _____

2.)
Ans.) In Flutter, everything is a widget - buttons, text, images, layouts, and even entire app itself. These widgets are organized in a tree structure called the widget tree.

- Widget Tree: Represents the visual structure and hierarchy of the app UI. Each widget is a node in the tree, defining part of the UI. When the app runs, Flutter builds this tree and renders it on the screen.

- Widget composition: Flutter encourages building UI by combining small, reusable widgets into more complex ones. Instead of creating large, monolithic components, you compose smaller, widgets, each handling a specific responsibility.

Eg -

```
class MyApp extends StatelessWidget {
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
  appBar: AppBar(title),
    body: Column(
  children: [
    Text(),
```

Teacher's Sign.: _____

ElevatedButton(),
),
);
}
}

b) Commonly Used Widgets and Their Roles:

1. Structural Widgets:
1.) Scaffold: Provides a basic page structure with an app body, floating action button, etc.
2.) App Bar: A toolbar for titles, icons and actions.
3.) Drawer: A side navigation panel.

2. Layout Widgets:
1.) Container: Adds padding, margins, borders and backgrounds
2.) Row and Column: Arrange widgets horizontally and vertically.
3.) Stack: Overlaps widgets on top of each other.
4.) List View: Creates scrollable lists.

3. Display Widgets:
1.) Text: Displays static or dynamic text
2.) Image: Displays images from assets, network or memory.

Teacher's Sign.: _____

3.) Interactive Widgets:
- Elevated Button: A button with elevation
- Textfield: Input field for text entry
- Checkbox, Switch, Slider: Interactive UI elements.

```
MaterialApp
 └ Scaffold
    ├ AppBar
    │  └ Text
    └ Column
       ├ Text
       └ Image
```
(Widget Tree)

Q3:

Ans.a) State management is crucial in Flutter because it controls how data flows and how UI updates in response to changes. In Flutter, state refers to data that affects the appearance or behavior of a widget. Managing state correctly ensures that UI stays in sync with underlying data.

Why State Management is Important

- UI Reactivity: Automatically rebuild the UI when the state changes, providing a dynamic user experience.
- Performance Optimization: Efficiently rebuild

Teacher's Sign.:

only the necessary widgets / preventing unnecessary rebuilders.

1. Data Consistency : Maintain consistent data across different parts of the app even with complex UI structures.

2. Seperation of Concerns : Helps keep business logic seperate from UI to de making apps more maintainable and testable.

ss 6) Comparing State Management Approaches in Flutter :

1.) Approach - setState
Description - Built-in, simple way to manage state by calling setState to rebuild a widget.
Pros :
- Easy to implement
- Great for small, local state
Best for : Small apps, local state (eg toggling a button, updating text fields)

2.) Provider.
Description - Inherited widget - based approach to pass data down the widget tree.
Pros :
Lightweight and flexible

Teacher's Sign.:

Good for medium
Easy to learn and widely supported
Best for: Medium to large apps

3) Approach - Riverpod
Description: A more robust, compile safe state management solution that fixes Providers limitations.

Pros - No need for context lookups.
2. Supports auto-disposal
More scalable and test friendly

Best for: Complex, large-scale apps

* When to Use Each Approach
- setState:
  - For small, local state changes within a single widget.
  - Example: A counter app.
- Provider
  When you need to share state across multiple widgets.
  Example: Managing user login status, theme switching, global settings.
- Riverpod:
  - For large, complex apps with multiple independent states.

Teacher's Sign.

Ex :- E-commerce apps with cart, etc

Q4)

Ans :) Integrating Firebase with a Flutter App / Application

Steps to Integrate Firebase in Flutter

1.) Create a Firebase Project :
- Go to Firebase Console → Create a new project

2.) Add an App to Firebase :
- Choose platform.
- For Android : Download the google-services.json file and place it in android/app directory.

3.) Update Flutter Project
- Add necessary firebase packages in pubspec.yaml :
  firebase_core, firebase_auth, etc
- Run flutter pub get to install dependencies

4.) Initialize Firebase
- Add firebase initialization in main.dart :

```
import 'package:firebase_core/
firebase_core.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
```

Teacher's Sign.: _____

runApp(MyApp());

3. Configure Native Platforms:
   For Android: Update android/app/build.gradle with google services plugin

Benefits of Using Firebase :-
1) Fully Managed Backend
2) Real Time DataBase and Sync
3) Authentication and Security
4) Scalability
5) Cloud Functions

Ans 4) Commonly Used Firebase Services:
1) Firebase Authentication user login / signup or social logins
2. Cloud Firebase: NoSQL database with real-time synchronization.
3. Realtime Database: JSON based database that syncs data in real time.
4. Firebase Analytics: Tracks user interactions and app events.
5) Firebase Storage: Stores and serves user-generated content like

Teacher's Sign.:

images, videos, etc.

Data Synchronization in Firebase,

```
import 'package:cloud_firestore/cloud_firestore.dart';

StreamBuilder(
stream: FirebaseFirestore.instance.
collection('messages').snapshots(),
builder: (context, snapshot) {
if (!snapshot.hasData) return
CircularProgressIndicator();
var messages = snapshot.data!.docs;
return ListView.builder(
itemCount: messages.length,
itemBuilder: (context, index) {
return ListTile(
title: Text(messages[index]
['text']),
);
},
);
},
);
```

Snapshots() → Stream Real-time updates