Name: Sujal. S. Tekwani
Class: D15B
Roll no: 59

MAD Assignment-1

Q1(a) Explain the key features and advantages of using flutter for mobile app development.

Ans(a) ① Single codebase for multiple platforms – Write one codebase for both android and ios, reducing development effort and maintainance.
② Hot reload – Instantly see changes in the app without restarting, making development faster.
③ Fast performance – 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.

Advantages :-

→ Faster Development time – Hot reload and single codebase reduce development time significantly.
→ Cost effective – Since the same code runs on both android and ios, businesses save on development and maintainance cost.
→ Reduced performance issues – The app runs natively without relying on intermediate bridges like in react native reducing lag.

(b) Discuss how flutter framework differs from traditional approaches and why it has gained popularity in developer community.

Ans(b) ① Single codebase vs Separate Codebase:

→ Traditional approach – Developers need to write seperate code for android (Java / Kotlin) and ios (swift)
→ Flutter – Uses a single dart base codebase for both platforms reducing development time and effort.

② Rendering Engine vs Native UI Components:

→ Traditional approach – Relies on platform-native UI components which can lead to inconsistencies and performance issues.
→ Flutter – Uses the Skia rendering engine to draw everything from scratch ensuring a consistent UI across devices.

Flutter gained popularity because:-

① Faster development with hot reload.
② Businesses save time and resources by maintaining a single codebase for multiple platforms.
③ Since flutter does not relies on native components, the UI looks and behaves same across different OS versions.
④ Works well with Firebase, REST APIs, GraphQL, and other backend technologies simplifying full stack development.

Q2 (a) Describe the concept of widget tree in flutter. Explain how widget composition is used to build complex user interfaces.

Ans 2. Widget tree in flutter:-
In flutter, the widget tree is the fundamental structure that represents the UI of an application. It is a hierarchical arrangement of widgets where each widget defines a part of the UI. Flutter's 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 :-
It refers to building complex UIs by combining smaller, reusable widgets. Instead of creating large, monolithic UI components, Flutter encourages breaking the UI into smaller manageable widgets that can be nested with each other.

Example: class Profile card extends stateless widget {
final string nam;
final string image Url;
Profile card ({required this.name, required this.image url});
@override
Widget build (Build context context) {
return Card (
child. Column (
Children: [
Image. network (image url),
SizedBox (height: 10)
Text (name, style: Textstyle (fontsize: 20, fontweight:
Fontweight: bold,
]
}}

Benifits of widgets Composition :-
① Reusability - Small widgets can be reused in different parts of the app
② Maintainability - Breaking UI into small widgets makes it easier to debug and update.
③ Performance - Flutter efficiently rebuilds only the necessary part of widget tree.

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

Ans (b) ① Structural Widgets :-
These widgets act as the foundation for building the UI.

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

```
Example:. Material App (
            home: Scaffold (
            appBar: AppBar (title. Text ("Flutter widget tree")),
            body: Container (
                padding: EdgeInserts.all (16.0),
                child: Text ("Hello, Flutter!"),
            )
        ),
    );
```

② Input and Interaction Widgets :-

- Textfield - Accepts text input from users.
- Elevation Button - A button with elevation.
- Gesture Detector - Detects gestures like taps, swipes and long presses.

```
Example:. Column (
            children: [
            Textfield (decoration: InputDecoration (labelText: "Enter name"
```

```
ElevatedButton(
  onPressed: () {
    print("Button Pressed");
  },
  child: Text("Submit"),
  ),
 ],
 );
```

③ Display and styling widgets widgets :-

- Text - displays text on the screen.
- Image - shows images from assets network or memory.
- Icon - Displays icons.
- Card - A material design card with rounded corners and elevation.

Example:
```
Column(
  children: [
  Text("Welcome to flutter!", style: Textstyle(fontsize:24,
                                  fontweight: Fontweight.bold)),
  Image.network("https://flutter.dev/images/flutter-logo-
                                  sharing.png"),
  ],
 );
```

Q3(a) Discuss the importance of state management in flutter applications.

Ans (a) In flutter, state refers to data that 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 state – Small, UI specific state that doesn't affect the whole app.

② App with status – Data should shared across multiple widgets.

Importance of Widget state management:

→ Efficient UI updates – Flutter's UI is rebuilt whenever state changes. Efficient state management ensures that only necessary widgets are updated.

→ Code maintainability and scalability – Managing state properly makes the code modular, readable and scalable for larger applications.

→ Data Consistency and Synchronization – Proper state management ensures that data remains consistent across different screens and widgets.

(b) Compare and contrast the different state management approaches available in flutter, such as setstate, provider and riverpod. Provide scenarios where each approach is suitable.

Ans (b) setstate – Local state

Pros: Simple built in easy to use.

Cons: Not scalable causes

Best use Cases: Small UI updates (eg. toggle switch, counter)

Provider – App wide state.

Pros: Lightweight, recommended by flutter, efficient.

Cons: Boiler plate 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 provider's limitations, improves performance
Cons: Requires learning new concepts.
Best use case: Large app needing global state (eg. shopping cart, user sessions).

Scenarios for Each approach :-

→ Use Setstate when 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 or theme changes.
→ Use Riverpod when building a complex, scalable app ~~which~~ with global state management, like an e commerce app with cart management.

Q4(a) Explain the process of integrating Firebase with a flutter application. Discuss the benifits of using Firebase as a backend solution.

Ans(a) Firebase provides a powerful backend solution for flutter applications offering services like authentication, real time databases, cloud functions, storage and more.
Steps to integerate Firebase with flutter

Step 1 Create a firebase project
- Go to firebase console.
- Click on "Add project" and enter a project name.
- Configure Google analytics if needed, the click create.

Step 2. Register the flutter app with Firebase
- In the firebase project dashboard click "Add App" and select Android or ios based on your platform.
- For Android: Enter the android package name and download the google services.json file and place it in android/app/.
- For ios: Enter the ios Bundle identifier. Download the Google service-info.plist-file and place it in ios/Runner.

Step 3. Install firebase dependencies
Add firebase dependencies in pubspec.yaml
firebase_core
firebase_auth
cloud_firestore
Run: flutter pub get.

Step 4. Configure Firebase for Android.
<i> Open android/build.gradel and ensure the following classpath 'com.google.gms:google-services:4.3.10'
<ii> Open android/app/build.gradle and add at the bottom apply plugin: 'com.google.gms.google_services'.

Step 5. Initialize Firebase in flutter

```
void main() async {
Widget FlutterBinding.ensureInitialized();
await Firebase.initializeApp();
runApp(MyApp());
```

Benifits of using Firebase

① Easy to setup and scale – No need to manage backend infrastructure. Scales automatically based on usage.

② Authentication – Provides email / password, google, facebook and phone authentication. Seamless integration with firebase authentication.

③ Cloud storage – Secure file storage for images, videos, etc.

④ Push Notification (Firebase cloud Messaging) – Send real-time notification to user across different platforms.

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

Ans(b) Firebase provides a suite of backend services that simplify flutter app development.

① Firebase Authentication – Enables secure authentication using email / password, phone number 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.instance.collection('users').add({
'name': 'John Doe',
'email': 'johndoe@example.com',
});

③ Realtime Database - A realtime JSON based database that automatically updates data across devices.
eg. ~~Database~~ DatabaseReference ref = FirebaseDatabase.instance.ref("message");
ref.set({"text": "Hello, Firebase!"});

④ Firebase Hosting - Deploys and serves web applications securely with automatic SSL.

- Data Synchronization in Firebase :-
Firebase ensures real-time data synchronization across multiple devices and platforms using Firestore and realtime database.

① Cloud Firestore Sync Mechanism - Uses realtime listeners to update UI instantly when data changes.
Eg. FirebaseFirestore.instance.collection('users').snapshots().listen((snapshot){
for (var doc in snapshot.docs){
print(doc['name']);
}
});

② Realtime Database Sync Mechanism - Uses persistent websocket connections for live updates.
Eg. DatabaseReference ref = FirebaseDatabase.instance.ref("messages");
ref.onValue.listen((event){
print(event.snapshot.value);
});

③ Offline Data Sync - Firestore caches data locally and syncs changes when device is online.
Eg. FirebaseFirestore.instance.settings = Settings(persistenceEnabled: true);