Aarya Chandu
D15B/14

MP2 Assignment No.1

04 M

Q1
a) Explain the key features and advantages of using Flutter for mobile app development

-> i) Single codebase for multiple platforms. Write one code base for both android and ios, reducing development effort and maintenance.
ii) Hot reload instantly see changes in the app without restarting making development faster and more interactive.
iii) Fast performance: Uses the Dart language and a compiled approach for smooth and high performance.

Advantages:-
i) Cost effective: Since the same code runs on both android and ios, business save on development and maintenance cost.
ii) Reduced performance issues: The app runs natively without relying on intermediate bridges like in react native reducing lag.
iii) Faster development: Hot reload and single code base reduce development time significantly.

b) Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the development community.

-> i) Single codebase vs seperate codebase
traditional approach developers need to write seperate code by android (kotlin/java) and iOS (swift).
Flutter uses a simple single dart based codebase for both platforms.

ii) Rendering Engine Vs Native UI components
Traditional approach: Relies on platform native UI component which can lead to inconsistencies and performance issues.
Flutter uses the skia rendering engine to draw everything from scratch.

Why flutter has gained popularity?
1. Faster development with hot reload. Developers can instantly see changes without restarting the app making the iteration process much quicker.
2. Cross platform efficiency saves time and resources by maintaining a single codebase for multiple platforms.
3. Consistent UI across devices, since flutter does not rely on native components, the UI looks and behaves the across different or versions.
4. Improved performance: AOT compilation and direct access to GPU rendering ensure smooth animations and high performance.

Q2 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 a hierarchical arrangement of widgets where each widget defines a part of the user which can be stateless or stateful.

MAD Assignment 1

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's by combining smaller, reusable widgets. Instead of creating large, monolithic UI components, Flutter encourages breaking the UI into smaller manageable widgets that can be reused and nested with each other.

Eg :-
```
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, fontwe-
      ight :20)
    );
```

Benefits of widgets composition

1. Reusability small widgets can be reused in different parts of the app
2. maintainability : Breaking UI into smaller widgets.

3. Performance: Flutter efficiently rebuilds only the ne parts of the widget tree.

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

-> 1. Structural widgets
These widget act as the foundation for building UI.
- MaterialApp: The root widget of Flutter app provides essential configurations
- Scaffold: Provides a basic layout structure including padding an app bar, body floating action button etc.
- Container: A versatile widget used for styling, pad margin and background customization.

Eg. MaterialApp (
home: Scaffold (
app.Bar: AppBar (title: Text ('Flutter widget tre
body: Container (
padding: EdgeInsets.all (16.0),
child: Text ("Hello, Flutter"),
),),);

2. Inputs & Interactions widgets
TextField - Accepts text input from users
Elevation Button - A button with elevation.
Gesture Detector - Detects gestures like taps, swipes lon presses.

MPL Assignment - 1.

Eg.

```
Column(
children: [
  TextField(decoration: InputDecoration(labelText: "Enter name"));
  ElevatedButton(
    onPressed: (){
    print("Button Pressed");
    },
    child: Text('Submit'),
    ),
    ],);
```

3. Display and styling 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

Ex.

```
Column(
  children:[
  Text("welcome to Flutter", style: TextStyle(fontSize: 24,
      FontWeight: FontWeight.bold)),
  Image.network("https://flutter.dev/images/flutter_
      logo_shortly.png");
  ],
  );
```

Q3

a. Discuss the important state management in flutter applicatio

-> In flutter state refers to data that can change during lifetime of an application. This includes:

- User inputs
- UI changes
- Network changes
- Animation states.

There are two types of states.

1. Ephimeral state: small, UI specific state that doesn't affect the whole app
2. App wide state: Data shared across multiple widge

Importance of state management.

- Efficient UI updates: Flutter UI rebuilt whenever state changes. Efficient state management ensures that on necessary widgets are updated, improving performanc
- Code Maintability and scalability: Manage state properly makes the code modular. Readable and scalable for large applications.
- Data Consistency and synchronization:- proper state m ensures that data remains consistent across different scens and widgets.

b. Compare and contrast the different statemanagement approaches available in flutter, such as setState, Provider and Reverpod. Provide scenarios where each approach is suitable.

~~Aditya Raj~~
~~D15B~~
~~01~~

MPL Assignment 1

setstate: Local state
pros:- simple built-in easy to use
cons: Not scalable causes unnecessary re-renders
Best use cases :- small UI updates (e.g. toggle switches)
~~Provides~~
Provider :- App wide state
pros - lightweight code for nested providers
cons - Boiler plate code for nested providers.
best use case - Medium scale apps.

Riverpod App wide state
pros- Eliminates providers limitations, improves performance.
cons- Requires learning new concepts.
Best use cases: Large apps needing global state.

Scenerios ~~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 frame changes.
- User Riverpod when building a complex, scalable app with global state management, like an ecommerce app with cart management.

Q4

a. Explain the process of integrating Firebase with Flutter application. Discuss the benefits of using Firebase as a backend solution.

-) Firebase provides a powerful backend solution for Flutter applications offering services like authentication, real time database, cloud functions, storage and more.

Steps to integrate Firebase with Flutter:

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

Step 2:
- Register the flutter app with firebase.
- In firebase project dashboard click "Add app" and select android or ios based on your platform
- for android: Enter the package name and download the google services.json file and places it in android/app/.
- for ios
Enter the ios bundle identifier.
Download the GoogleService-Info file and place it in ios.

Step 3
- Install firebase dependencies
- Add firebase dependencies in pubspec.yaml.
- Firebase auth.
- Run flutter pub get.

MPL assignment 1

Configure Firebase for android & ios

For android

1. open android/build.gradle and ensure the following classpath 'com.google.gms:google-services:4.3.10'
2. open android/app/build.gradle and add at the bottom.

Steps

Initialize Firebase in Flutter

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

Benefits of using Firebase

Firebase is a backend as a service (BaaS) that simplifies backend development for Flutter apps. Here are some key benefits.

1. Easy to setup & 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, vides and documents.

6. Highlight the Firebase services commonly used in Flutter development and provide a brief o how data synchronization is achieved.

-) Firebase provides a suite of backend servic that simplify Flutter development.

1. Firebase Authentication
Enables secure authentication using email/password, phone no. and third party providers like google, Facebook and Apple.

2. Cloud Firestore
Stores and sync data in real time across devices supports structure data, queries and offline access.

e.g 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.

e.g DatabaseReference ref = FirebaseDatabase.instanc
ref('message')

4. Firebase cloud Messaging
Enables push notifications and messaging b/w devs

e.g FirebaseMessaging.instance.subscribeToTopic('news

MPL Assignment 1

5. Firebase Hosting
Deploys and serves web applications securely with automatic.

Data synchronization in Firebase
Firebase ensures real-time data synchronization across multiple devices and platforms using ~~Firest~~ 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 w
the device is online.

eg Firebase.firestore().instance.settings = settings (persistenc
Enabled: true);