Aaryan Kalbhar
D15B - 26

AH 04

MAD Assignment - 1

Q1)

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 maintainence.

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

3 Fast Performance : Uses the Dart language and a compiled approach for smooth and high performance apps

4. Open Source & 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 codebase reduces development time significantly.

(2) Cost effective : Since the same code runs on both Android and iOS, business save on development and maintainence cost.

(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 traditional approaches and why it has gained popularity in the developer community.

Ans 1) Single codebase vs separate codebase

- Traditional Approach : Developers need to write separated code for Android (Java / Kotlin) and iOS (Swift)
- Flutter : Uses a single dart based codebase for both platforms reducing development time and effort.

2) 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 ensuring a consistent UI across devices

Why flutter has gained popularity.

① Faster development with Hot Reload. Developers can instantly see UI changes without restarting the app making the iteration process much quicker.

② Cross platform Efficiency : Business save 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 same across different OS versions

(4) Improved performance : AOT compilation and direct access to GPO rendering ensure smooth animations and high performance

(5) Easy Integration with Backend Technologies : works well with Firebase, REST API's GraphQL, and other backend technologies simplifying full stack development.

Q3)

a. Describe the concept of widget tree in flutter. Explain how widget composition is used to build wear interfaces.

Ans widget tree in flutter.

In flutter, the widget tree is the fundamental structure that represent the UI of an application. It is a hierachial arrangement of widgets where each widget define a part of user interface. 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

Widget composition 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 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,
fontWeight: FontWeight.bold))
]
);
}
}

Benefits of widgets composition

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

b. Provide example of commonly used widgets and their role in creating a widget tree.

→ 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 configurations
- Scaffold – provides a basic layout structure, including an app for body floating action button etc
- container :- A versatile widget used for styling, padding, margin & background customization

Eg:- MaterialApp()

```
home : Scaffold (
    appBar : AppBar ( title : Text ('Flutter widget Tree')),
    body : Container (
        padding : EdgeInsets.all (16.0),
        child : Text ("Hello");
    )
)
```

② Input & Interaction widgets

TextField - Accepts text input from users

ElevationButton - A button with elevation

GestureDetector - Detects gesture like taps, swipes and long presses.

eg:- Column (
children: [
TextField (decoration: InputDecoration (labelText: "N
ElevatedButton (
onPressed: () {
print ("Button pressed");
},
child: Text ("submit"),
),
],
);

(3) Display & Styling widget

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

eg: Column (
children: [
Text ("Welcome", 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

→ In flutter, state refers to data that can change during the lifetime of application. This include:
- User input
- UI changes
- Network changes
- Animation states

There are two types of state
(1) Ephemeral State: Small, UI specific state that doesn't affect the whole app.
(2) App wide state - Data shared across multiple widgets. ~~Impor~~

* Importance of state management:
- Efficient UI update: Flutter's UI is rebuilt whenever state changes. Efficient state management ensure that only necessary widgets are updated.
- Code Maintainability & Scalability: Managing state properly makes the code modular, readable and scalable for larger application
- Data consistency and synchronization
Proper state management ensures that data remain consistant across diff screens & widgets.

b Compare and contrast the diff state management approaches available in flutter, such as setState Provider, and Riverpod provide scenarios where each approach is suitable.

→ setState - Local state
Pros - simple built in easy to use
Cons - Not scalable causes unnecessary re-renders
Best-Use Cases - Small UI updates (eg, counter, toggle switch)

Provider - Appwide State
Pros - lightweight, recommended by flutter, efficient
Cons - Boilerplate code for nested providers
Best-Use Cases - Medium Scal App (eg. Authentication, theme, API data)

Riverpod : Appwide State (More Scalable than provider)
Pros : Eliminates provider's limitations, Improves performance
Cons - Requires learning new concepts
Best usecases - Large apps needing global state (eg shopping cart, user sessions)

Scenarios for Each Approach
1. Use setState when managing simple UI elements with in a single widget
2. Use Provider when sharing state across multiple widgets
3. Use Riverpod when building a complex, scalable app with global state management

(14)

a) Explain the proces of integrating firebase with a flutter application. Discuss the benefits of using firebase as backend solution.

→ Firebase provide a powerful backend solution for flutter application offering servies like authentication real time database, cloud functions, storage and more.

* Steps to Integrate.

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

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 Android package name and download the google servie .json file and place it in android/app/
- For ios: Enter the ios bundle identifies Download the GoogleService-Info.plist-file and place it in ios/runner

③ Install firebase dependineses
Add firebase dependencies in pubspec.yml.
firebase core
firebase auth
cloud firestore, Run: flutter pub get

④ Configure Firebase for Android & iOS

for Android

1) Open Android/build.gradel and ensure the following classpath 'com.google.gms:google-services:4.3.10'

2) Open android/app/build.gradle and add the bottom apply plugin: 'com.google.gms.google-servi

⑤ Initialise Firebase in flutter

```
void main() async {
  widget FlutterBinding.ensureInitialised();
  await Firebase.initialiceApp();
  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 set up & Scale

No need to manage backend infrastructure

Scale 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 img, videos & documents

4) Push Notification (Firebase cloud Messaging)

and real-time notification to user across different platform

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

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

Eg:- Firebase.firestore.instance.collection('users').add({
'name': 'Aaryan',
'email': 'test@gmail.com',
});

3) Realtime Database
A realtime, json based database that automatically update data across devices

ex. DatabaseReference ref = FirebaseDatabase.instance.ref("message");
ref.set({"text": "Hello."});

4) Firebase Cloud Messaging (FCM)
Enables push notifications and messaging between users

ex. FirebaseMessaging.instance.subscribeToTopic("news");

5) Firebase Analytics

Tracks user interaction and app performance

Ex: Firebase Analytics analytics. firebase Analytic. Instance analytics. log Event (name: "button - clicked", parameters: { "button": "subscribe" });

6) Firebase Hosting

Deploys and serves web application securely with automate 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

Used realtime listeners to update UI instantly when data changes

Eg:- Firebase Firestore . instances . collection ('users'). snapshots(). listen ((Snapshot) {

for (var doc in snapshot. docs){

print (doc ['name']);

}

});

2 Realtime Database Sync Mechanism
Uses persistent WebSocket connection for live update.
Ex. DatabaseReference ref = FirebaseDatabase.instance.ref("messages");

```
ref.onValue.listen((event){
  print(event.snapshot.value);
});
```

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

4 Cloud Functions for automated updates
Automate backend logic to trigger updates when data changes.