Name: Jayan. Karkera Div = D15B Roll no: 28

MAD

Assignment -01. AH

DATE:

Q1) a) Explain the Key features and advantages of using flutter for mobile app development?

→ 1) Single code base for multiple platforms
2) Instantly see changes in the app without restarting making development faster and more.
3) Fast performance by using Dart language and compiled approach for smooth and high performance apps.
4) Open Source & Strong community support.
5) Backed by Google and large developer community.

Advantage:
1) Faster Development Time
2) Cost effective
3) Reduced performance issues.

Q1) b) Discuss how flutter framework differs from traditional approaches and why it has gained popularity in the developer community.

→ 1) Single codebase vs seperate codebase.
→ Traditional approach: Developer needs to write seperate code by Android (Java/Kotlin) and ios (Swift)
→ Flutter uses single dart based codebase for both platforms reducing development time and effort.

2) Rendering Engine vs Native UI component.
→ Traditional approach: Relies on platform native UI component which can lead to inconsistencies and performance issues.
→ Flutter: uses the Skia rendering engine to draw every thing, from scratch ensuring a consistent UI across devices.

DATE:

Q1) Why Flutter has gained popularity?

1) Faster development with Hot Reload: Developer can instantly see UI changes without restarting the ap making the iteration process much quicker.

2) Cross Platform Efficiency: Businesses save time and reso by maintaining a single codebase for multiple platfo

3) Consistent UI across devices: Since Flutter does not r on native components, the UI looks and behaves the same across different OS versions.

4) Improved performance: AOT compilation and direct accesses to GPU rendering ensures smooth animation and high performance.

5) Easy Integration with Backend Technologies: Works wi with firebase, REST Api, GraphQL, and other backend technology 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?

→ Widget tree in Flutter:

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

Widget composition in Flutter: ~~tree determine how the~~

Widget composition refers to building complex UIs by combining smaller, reusable widgets. Instead of creating large, monolithic UI components, Flutter encourage

~~tree~~

DATE:

breaking the UI to smaller manageble widget that can be reused and nested with each other.

example: class profilecard extends StatelessWidgets{
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,
fontWeidth: 7)
]
);
}

Benefits of widget Composition.

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

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

i) Structural widget
These widget act as the foundation for building UI.
Material App: The root widget of a flutter app that provides essential configuration.

- Scaffold - Provides a basic layout structure, including app bar, body, floating action button etc.
- Container - A versatile widget used for styling, padding, margin and background customization.

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

2) Inputs & Interaction widget

TextField - Accepts text input from users.

ElevationButton - A Button with elevation

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

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

DATE:

Display & setting 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,));
Image.network ("https://flutter.dev/image/flutter-logo-sharing.png"),
];
);

a) Discuss the importance of self management in flutter application

→ In flutter state refers to data that can change during the lifetime of an application. This includes:
- User Input, UI changes, Network change, Animations state.

There are two types of states.

1) Ephemeral State: Small UI specific state that does not effect the whole app.

2) App wide state - Data shared across multiple widget

Importance of state Management:
- Efficient UI updates
- Code maintainability.
- Data consistency & Synchronization.

DATE:

Q3) b) Compare and contrast 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: 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 - Elimination Providers limitations, improves performance

Cons - Require learning new concepts

Best use cases: Large Apps nuding global state (eg: shopping cart, user session)

Scenario for each approach.

- Use setstate when managing simple UI elements with single widget like toggling dark mode in setting.
- Use provider when sharing state across multiple widgets such as managing user authentication or theme changes.
- Use Riverpod when building a complex, scalable app with global state management like e-commerce app with cart management.

DATE:

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

Firebase provides a powerful backend solⁿ. by Flutter app offering services like authentication, real time database, cloud functions, storage and more

Steps to integrate Firebase

Create a firebase project.
- Go to firebase console
- Click on "Add Project" and enter a project name.
- Configure ~~Toggle~~ analytics if needed then create.

Register the Flutter app with Firebase
- click Add app and select Android or ios
- for android: add android package name.
- for ios: Enter ios Bundle Identifier.

Install Firebase dependencies
- ~~Add~~ Fire base dependency in ~~pubspec~~ pubsec.yaml
- firebase core
- firebase auth
- cloud firestor.
- ~~Run~~ Run flutter pub get

Configure Firebase for Android or ios.
For Android
change in build.gradel and chang some of the dependencies

Initialize Firebase in Flutter

DATE:

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

...

Benefits of using Firebase. (Backend as Services)

1) Easy set up and scale.
2) Authentication.
3) Cloud storage
4) Push notification.

...

Qu) b) Highlight the firebase commonly used in flutter dev provide a brief overview of how data syn is achie

→ firebase provides a suite of backend services that simpli flutter app development.

1) Firebase Authentication
Enables secure authentication using email/phone no and p and third party like google, facebook and apple.

2) Cloud Firebase.
Stores and syncs data in real time across devices sup structured data, queries and offline acces

3) Real time Database
A real-time, JSON-base database that automatically updates date across devices

4) Firebase cloud Messaging (FCM)
Enable push notification and messaging b/w users

Data Synchronization in firebase

→ firebase ensures real-time data syn across multiple devices and platforms using Firebase and realtime Database cloud firestor syn mechanism

uses realtime 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']);
}
});

Realtime Database Syn Mechanism

Uses persistent websocket connections for live updates.

Ex = DatabaseRef = FirebaseDatabase.instance.ref('messages');
ref.onValue.listen((event) {
print(event.snapshot.value);
});

8) offline Data Sync.

Firestore caches data locally and syncs changes when the device is online

Ex: Firebase.firestore.instance.setting.(persistence Enabled: true);

4) Cloud function by automated updates

Automates backend logic to trigger updates when data changes.