NAME:_____ STD.:_____ DIV.:_____ DATE: PAGE:

Akshad Mhaskar
D15B / 32

MPL Assignment 1

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

→ Key features:

1) Single Codebase - Write one codebase for andriod & ios, reducing development & effort & maintainenance
2) Hot Reload - Instantly see changes in the app without restarting making development faster & more interactive
3) Fast Performance - Uses the Dart language & a compiled approach for smooth & high performance apps
4) Open Source & strong community support: Backed by Google & a large developer community, ensuring continuous improvements & resources.

Advantages.

1) Faster Development time: Hot reload & single codebase reduce development time significantly
2) Cost effective: Since the code runs on both Andriod & ios, business save on development & maintainence cost.
3) Reduce Performance issues: The app runs natively without relying on intermediate bridges like in react native reducing lag.

b) Discuss how the flutter framework differs from traditional approaches & why it has gained popularity

→ How Flutter Differs from traditional Approaches

NAME:_____ STD.:_____ DIV.:_____ DATE: PAGE:

1) Single codebase - Traditional methods need separate code for android & ios, but flutter uses one code for both.
2) Hot Reload - Traditional apps require full restart after changes, but flutter updates instantly.
3) UI Rendering - Traditional apps ~~require~~ use native components, while flutter has its own rendering engine (Skia) for faster performance
4) Customization - Traditional UI design depends on platform specific components, but flutter provides fully customizable widgets.

Why Flutter has gained popularity

1) Faster development with hot reload: Developers can instantly see UI changes without restarting the app making iteration process much quicker.
2) Cross Platform Efficiency: Businesses save time & resources by maintaining a single codebase for multiple platform.
3) consistent UI Across devices: Since flutter does not rely on native components, the UI looks & behaves the same across different OS versions.
4) Improved performance: AOT compilation & direct accesses to GPU rendering ensure smooth animations & high performance.

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

NAME:_____ STD.:_____ DIV.:_____ DATE : PAGE :

→ 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 interface Flutter is UI is entirely built using widgets which can be stateless or stateful. The widget tree determines how the UI is rendered & updated when changes occur.

Widget Composition in Flutter

Widget composition refers to building complex UI by combining smaller, reusable UI widgets. Instead of creating large, monolithic UI components, flutter encourages breaking the UI into smaller manageable widgets that can be reused & nested with each other.

Benefits of widget composition

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

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

→ 1) structural widgets

These widgets act as the foundation building the UI

NAME:_____ STD.:_____ DIV.:_____ DATE : PAGE :

- Material App: the root widgets of a flutter app that provid essential configuration.
- Scaffold: Provides a basic layout structure, including an app bar, body Floating action button etc.
- Container - A versatile widget used for styling, padding, margin & background customization.

Ex :- MaterialApp(
home : Scaffold(
appBar : AppBar(title : Text("flutter widget tree"
body : container(
padding : EdgeInserts.all(16.0),
child : Text("Hello, Flutter !"),
),
),
);

⇒ 2) Input & Interaction Widgets
Textfold : Accepts text input from user.
Elevation Button - A button with elevation.
GestureDetector - Detects gesture like taps, swipes & long presses.

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

NAME: STD.: DIV.: DATE: PAGE:

3) Display & Styling Widgets
- Text - displays text on the screen
- Image - shows images from assets network/memory
- Icon - Displays icon
- Card - A material design & with card with rounded corners & elevation.

Ex :-
```
column(
  children: [
  Text("Welcome to flutter!", style: Textstyle(Fontsize:20
                                  Fontweight: Fontweight.bold)),
  Image.Network("https://flutter.dev/images/flutter-logo
                                  -sharing.png"),
  ],),
```

Q3) a) Discuss the important state managment in flutter applications.

→ 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) ~~Epheral~~ Ephemeral state: small, UI specific state that doesn't affect the whole app.

2) App wide status - Data shared across multiple widgets

- Importance of State Mgmt
- Efficient UI updates: Flutter's UI is rebuilt whenever state changes. Efficient state mgmt ensures that only necessary widgets are updated improving performance.

NAME: STD.: DIV.: DATE: PAGE:

- Code Maintainability & Scalability : Managing state properly makes the code modular, readable & scalable for larger applications.
- Data Consistency & Synchronization - Proper state mgmt ensures that data remains consistent across different screens & widgets.

b) Compare & constrast the different state mgmt approaches available in flutter, such as setstate, Provider & Riverpod. Provide scenarios where each ~~components~~ approach is suitable.

→ setstate - local state
Pro - Simple built in easy to use
Cons - Not scalable causes unnecessary re-renders
Best use cases - Small UI updates (eg:- counter)
Provider - App wide state
Pros - Lightweight, recommended by flutter
Cons - Boilerplate code for nested providers.
Best use case - Medium Scale App (eg:- API data)
Riverpod : App-wide state (More scalable)
Pros : Eliminates Provider's limitation, improves performance
Cons : Requires learning new concepts
Best use cases : large apps needing global state (eg :- Shopping cart)

Scenar

Scenarios for each Approach

- Use setstate when managing simple UI elements with in 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 with cart mgmt.

Q4) a) Explain the process of integrating firebase with a flutter application. Discuss the benefits of using firebase as a backend solution.

→ Integrating Firebase with a Flutter Application

Step 1: Create a Firebase project

1. Go to Firebase console & create a project
2. Download the config (google-services.json for Andriod).

Step 2: Add Firebase Dependencies

In pubspec.yaml:

dependencies:
firebase_core: latest_version
firebase_auth: latest-version
cloud_firestore: latest-version

Run flutter pub get to install dependencies.

Step 3: Initialize firebase.

Modify main.dart.

```
import 'package:firebase_core/firebase_core.dart';
void main() asyn {
```

```
WidgetsFlutterBinding.ensureInitialized();
await Firebase.initializeApp();
  runApp(MyApp());
}
```

Step 4: Use Firebase service
Authentication: firebase_auth for login/signup
Database: cloud_firestore for real-time data.
Storage: firebase_storage for file uploads
Notifications: firebase_messaging for push notification.

Benefits of using firebase

1. Serverless Backend - No need to manage servers.
2. Real-Time Data Sync - Firestore enables instant updates.
3. Scalable & Secure - Google-managed infrastructure
4. Easy authentication - Supports google, facebook, email login.
5. Cloud functions - Automate tasks with serverless function.
6. Push notifications - free messaging via FCM.
7. Cost-Effective - Free tier with pay-as-you-go pricing.
8. Analytics & crash Reports - Firebase Analytics & Crashlytics.

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

→ Common Firebase services in flutter

1. Firebase Authentication - Secure login via email, google, Facebok or phone.
2. Cloud firestore - NoSQL database with real-time syncing.
3. Realtime Database - low-latency database for instant updates.
4. Firebase Storage - stores & retrieves files like images & video.
5. Firebase cloud Messaging (FCM) - ~~Enab~~ Enables push notifications.
6. Firebase crashlytics - Tracks & reports app crashes.
7. Firebase Cloud functions - Runs backend tasks serverlessly.
8. Firebase Analytics - Provides insights into user behaviour

Data Synchronization in Firebase.

1. Realtime Listeners - Apps receive instant updates when data changes.

Ex:-

```
FirebaseFirestore.instance.collection('users').
snapshots().listen((snapshot){
  for (var doc in snapshot.docs){
    print(doc.data());
  }});
```

2. offline persistence - Caches data for offline access & syncs when online.
3. conflict Resolution - Uses timestamps to maintain data consistency.

Firebase's real-time sync & offline support make it ideal for chat apps, live updates & collaborative tools in flutter.