Sanika Mehetre
D15B - 34

MPL Assignment - 1 (AH) (05)

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

⇒ 1) Single codebase for multiple platforms: Write one codebase for both Android and iOS, reducing development effort and maintenance.

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 app.

4) Open source & strong community support: Backed by Google and a large developer community, ensuring continuous improvements and resources.

Advantages:

i) Faster Development Time: Hot Reload and single codebase reduce development time significantly.

ii) Cost effective: Since the same code runs on both Android and iOS, businesses save on development and maintenance cost.

iii) Reduced 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 and why it has gained popularity in the development community.

⇒ 1.] Single codebase V/S separate codebase:
- Traditional Approach: Developers need to write separate 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 V/S 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.

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: Businesses 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 accesses to GPU rendering ensure smooth animations and high performance
5) Easy Integration with Backend Technologies: Works well with Firebase, REST API, GraphQL and other backend technologies.

Q2.a) 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 represented the UI of an application. It is a hierarchical arrangement of widgets where each widget defines a part of the user interface.

Widget composition in flutter:

Widget composition refers to building complex UIs by combining smaller, reusable widgets. Instead of creating large,

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

3) Display & Styling Widgets

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

```
eg: 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 important state of management in flutter applications.

⇒ In Flutter, state refers to data that can change during the lifetime of an application.

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 bar, body floating action button etc.
- container: A versatile widget used for styling, padding, margin and background customization.

```
eg: 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:

Textfield - accepts text input from users.
ElevationButton - A button with elevation.
GestureDetector - Detects gestures like taps, swipes and long presses

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

monolithic UI components.

Example:

```
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) Maintainability: Breaking UI into smaller widgets makes it easier to debug and update.

Q2. b) Provide examples of commonly used widgets and their roles in creating a widget tree.

⇒ 1) Structural widgets:

These widgets act as the foundation

Step 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.

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.

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

Provider - App wide state

Pros: Lightweight, recommended by flutter, efficient.

Cons: Boilerplate code for nested providers

Best use case: Medium Scale apps (eg: authentication, themes, API data)

Riverpod App - Wide state (more scalable than provider)

Pros: Eliminates Providers limitations, improves performance

Cons: Requires learning new concepts

Best use case: Large apps needing global state (eg: shopping cart, user sessions)

Q4.a) 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 solution for flutter applications offering services like authentication, real time databases, cloud functions, storage and more.

Steps to integrate Firebase with flutter

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 wide status: Data shared across multiple widgets

Importance of State Management:
- Effecient UI updates: Flutter's UI is rebuilt whenever state changes
- Code Maintainability & Scalability: Managing state properly makes the code modular, readable and scalable for larger applicatio

Q3.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.

⇒ Setstate - Local state

Pros - Simple built in easy to use

Cons - Not scalable causes unnecessary re-renders

Best use cases - Small UI updates

4) Firebase Cloud Messaging (FCM)
Enables push notifications and messaging between users
eg: FirebaseMessaging.instance.subscribeToTopic("news");

5) Firebase Analytics:
Tracks user interaction and app performance.
eg: FirebaseAnalytics.analytics.FirebaseAnalytics.instance;
analytics.logEvent(name: "button clicked", parameters: {"button", "subscribe"});

6) 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 Firebase and realtime Database.

1) Cloud Firestore Sync Mechanism:
Uses real time listeners to update UI instantly when data changes

2) Realtime Database Sync mechanism:
Uses persistent websocket connections for live updates.

3) Offline Data Sync:
Firestore caches data locally and syncs chang

Q4.b) Highlight the firebase services 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.

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': 'John Doe',
'email': 'johndoe@example.com',
});

3) Realtime Database:
A realtime, json-based database that automatically updates data across devices.

eg: DatabaseReference ref = FirebaseDatabase.instance.
ref("message"),
ref.set({"text": "Hello, Firebase!"});

Step 4: Configure Firebase for Android & iOS

For Android:

1) Open android / build.gradle and ensure the following classpath 'com.google-services:4.3.10'
2) 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 {
  WidgetFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
    runApp(MyApp());
```

Benefits of using Firebase:

1) Easy to setup & scale: No need to manage backend infrastructure.
2) Authentication: No need to manage Provides emails, password.
3) Cloud Storage: Secure file storage for images, videos and documents.
4) Push Notifications: Send real-time notifications to user across different platforms.