Aakanksha. Suvarna
D15B
65

Page No.
Date

MAD & PWA Lab

Assignment 1
LO Mapping: 1, 2, 3

Q1]

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

Ans→ The key advantages of using flutter include customisable widgets, cross platform capabilities and strong community support. Other advantages include

1. Single Codebase: Develop for ios and android from a unified codebase reducing time & effort.

2. Rich widget library: Pre-designed, customizable widgets for consistent and visually appealing UI.

3. High performance: flutter ensures smooth performance

4. Consistent UI Across platforms: flutter adapts to the design principles of each platform

5. Cost-effective: reduces development cost with a single codebase and efficient development process

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

Page No.
Date

ning

Ans→ I) Differences from traditional approaches

1. Single codebase: Flutter uses single codebase, while traditional approaches require seperate code bases.

2. Widget-based UI: Traditional approaches rely on Platform-specific components

3. Dart language: Flutter employees dart, a language specific to the framework, different from the platform-specific languages used in traditional approaches

II) Reasons for popularity

1. Efficiency & Time saving: Flutter reduces development time by enabling code reuse for multiple platforms.

2. Consistent UI across platforms

3. Rich widget library

4. Strong community support: Flutter's supportive community fosters a growing ecosystem

5. Cost effectiveness: due to streamlined development process

6. Access to native features:

Page No.
Date

Q2)

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

Ans→ In Flutter, the widget tree is a heirarchical representation of user interface components, where each node corresponds to a widget defining the structure and appearance of the UI. Widget serve as the fundamental building blocks, ranging from basic elements like buttons and text to more complex structures.

Widget composition is a core concept in Flutter allowing developers to build intricate user interfaces through the assembly of simple & reusable widgets.

Developer start with foundational widgets and progressively compose them into more sophisticated structure

Custom widgets can be created by encapsulating functionalities or extending Flutter's stateful Widgets or 'Stateless Widget' classes promoting reusability.

The heirarchical arrangement of widgets in the tree mirrors the layout and composition of the UI. Widget can be nested, allowing for the creation of complex interfaces. This enhances code reusability, maintainability.

In summary, the widget tree and composition in flutter provide a structured and flexible framework for developers to efficiently construct, customize and maintain complex UI.

Ans.

Page No.
Date

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

1) Container widget-

Role: A versatile container that can hold & decorate other widgets.

Example:

```
Widget build (BuildContext context) {
  return Container (
    child: Text ('Hello, Flutter!'),
  );
}
```

2) Column & Row widget

Role: Organise child widgets vertically (Column) or horizontally (Row)

Example:

```
Widget build (BuildContext context) {
  return Column (
    children: [
      Text ('Item 1'),
      Text ('Item 2'),
    ]
  );
}
```

Page No.
Date

3) List View Widget

Role: Creates a scrollable list of widgets

Example

```
Widget build (Build Context context) {
  return ListView (
    children: [
      List Tile (title: Text ('Item 1')),
      List Tile (title: Text ('Item 2')),
    ]
  );
}
```

4) AppBar Widget.

Role: Represents the app bar at the top of the screen

Example:

```
Widget build (Build Context context) {
  return Scaffold: (
    appBar: AppBar (
      title: Text ('My App'),
    ),
    body: // other widgets go here
  );
}
```

5) Image Widget

Role: Displays images in the UI

Example

```
Widget build (BuildContext context) {
    return Image.asset('assets/my_image.png');
}
```

Q3)

a) Discuss the importance of state management in Flutter applications

Ans→ 1) Dynamic UI:

State management is critical for handling dynamic changes in UI. Whether it's updating UI elements in response to user interactions or reflecting changes in data, effective state management ensures that UI remains responsive and reflects the current application state

2) Code reusability:

Well managed state enables the creation of modular and reusable components. In Flutter, where widgets can be composed and reused, it ensures that these components can be easily integrated into different parts of the application.

Page No.
Date

3) Cross - Screen Communication:

State management facilitates communication between different screens or components of an application allowing them to share and synchronize data.

4) Maintainability and debugging

Clear and well-organised state management makes it easier to debug and maintain code.

5) Efficient Memory Usage

Effective state management helps optimize memory by ensuring that only the necessary components are rebuilt when state changes occur, preventing unnecessary widget rebuilds.

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

1. SetState

Description: The setState method is a built-in mechanism in flutter for managing the internal state of a stateful Widget.

Suitable Scenarios:

Simple UIs: setState is suitable for small to moderately complex UIs where state changes are localised to a specific widget and don't need to be shared across the entire application

2) Provider:

Description: The provider package is a popular and lightweight state management solution in flutter. It follows the provider pattern and is based on Inherited Widget.

Suitable Scenarios:

Scoped State: Provider is suitable for managing state within specific parts of the widget tree, creating a scoped and efficient solution.

3) Riverpod.

description: Riverpod is an advanced state management library and a successor to Provider. It provides a broader set of features and is designed to be more modular and testable

Suitable Scenarios.

Complex applications:

Riverpod is suitable for large and complex applications where a more structured and testable state management approach is needed.

Page No.
Date

Q4) a) Explain the process of Integrating firebase with Flutter application. Discuss the benefits of using firebase.

→ Integrating firebase with flutter:

- Create a firebase Project: Start by creating a project on the firebase console and configure your app.
- Add ~~flutter~~ firebase to flutter project: In your flutter project, add the necessary dependencies by updating the pubspec.yaml file:

```
yaml
dependencies:
  firebase_core: ^latest_version
  firebase_auth: ^latest_version
  cloud_firestore: ^latest_version
```

- Run flutter pub get to fetch the dependencies.

Initialize firebase: In your flutter app by calling firebase.initializeApp() in the main() method:

```
dart
import package:firebase_core/firebase_core.dart;
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

Page No.
Date

Use firebase services: like authentication, firebase or other in your flutter app by importing the relevant packages and initializing them using the firebase

- Handle firebase dependencies: Ensure proper error handling and dependency management when dealing with asynchronous firebase operations use try, catch blocks to handle exceptions

Benefits of using firebase

1) Real Time database (firestore)
   firebase provides cloud firestore, a real time NoSQL database, enabling seamless data synchronization across devices

2) Authentication: firebase authentication simplifies user authentication with various methods such as email | password, Google Sign-In etc

3) Cloud storage: firebase cloud storage provides scalable and secure file storage with easy integration into flutter applications

4) Easy integration: firebase integrates seamlessly with flutter, providing a range of SDKs and plugins that simplify backend development.

Page No.
Date

Q4]
b Highlight the firebase services commonly used in Flutter development and provide a brief of how data synchronisation is achieved

⇒ Firebase services commonly used in flutter

1) Firebase authentication: Provides secure user authentication using various methods such as email / password Google sign in and more. Allow developers to manage sign-ins, sign-outs and identity verification

2) Firebase Hosting: Provides secure and fast hosting for web apps, static content and microservices Integrates seamlessly with other firebase services

3) Firebase Analytics: Provides insights into user behaviour and app performance. Helps developers make data-driven decision and optimize user experiences

4) Firebase crashlytics: offers crash reporting to identify and prioritize stability issues in the app. Enables developers to resolve critical errors

5) Cloud firestore: Firestore activities real time data synchronization through the use of data listeners.