Shreyash D. Dhasade
D15B/14

Date
Page

MAD Assignment 1

Q.1) a) Explain the key features and disadvantages of using flutter for mobile app developement.

→ Flutter is a cross platform UI toolkit developed by google for building natively compiled applications for mobile, web and desktop from single codebase.
Key feature and advantages include:
1. Hot reload: Enables developers to instantly view changes without restarting.
2. Widget based architechture:
UI-components in Flutter are widgets making the developement modular and customizable
3. Expressive UI: flutter provides a rich setup, customizable widgets for creating visually appealling interfaces.
4. Single codebase: Develope once, ~~develope~~ deploy everywhere, reducing developement time and effort.

5. Strong community support: A large and active community contributes to a wealth of resources and packages.

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

Teacher's Sign.: ____

Flutter uses reactive framework, whereas traditional approaches are typically imperative.

2. Flutters offers a consistent UI across platforms ensuring a native look and feel.

3. The use of dart language and widget based approach enhances developer productivity.

4. popularity arises from efficient developement process, performance, and the vibrant community.

Q.2) a) Describe the concept of the widget tree in flutter. Explain how widget composition is used to build complex composition.

→ 1. In flutter, the widget is a fundamental concept that represents the hierarchy of user interfaces elements in an application. Every in flutter is a widget, whether it's a button, text, image or even the entire application itself.

2. The widget tree is composed of various type of widgets, each serving a specific purpose. Widgets in flutter can be broadly categarized into two: stateless and statelful.

Date
Page

3. stateless widgets are immutable and don't have any internal state, while stateful widgets can change their internal state during their lifetime.

b) Provide examples of commonly used widgets.

→

1. Material app: defines the structure of a flutter app.

2. Scaffold: Represents the basic visual inter-face of the app, including the app bar and body.

3. Container: A box model that can contain other widget, providing layout and styling

4. Row and column: Arrange child and widget horizontally or vertically.

5. List view: Displays a scrolling list of widgets.
6. Floating action button: represents a floating action button.

Q.3) a) Discuss the importance of state management in flutter application

→ State management is a crucial aspect of building robust and efficient flutter application. In flutter "state" refers to data that influences

Teacher's Sign.:

appearances and behaviour of widgets. Managing state effectively is essential for creating responsive, dynamic and scalable applicat-ions. Here are some key reasons why state management is important in flutter.

1. User Interface updates
2. performance optimization
3. code manageability.
4. Reusability and and modularity.
5. persistence and navigation.
6. stateful widget navigation

b) Compare and contrast state management approaches in flutter, such as setstate provider, and riverpod, provide scenarios where each approach is suitable.

→ 1. setstate:

This is the most straight forward way to manage state in flutter. It is built into the framework and is easy to understand for beginners.

- Appropriate for simple UI's:

For small and moderately complex UI's where state changes are localized and widget free and the widget tree is not deeply nested set state can be sufficient.

Cons:

setstate is limited to the descendant

over rebuilding widgets: It triggers rebuild of an entire widget and its subtree, potent-ially, causing performance issues for large application.

Suitable scenarios:
- small moderately sized applications
- simple UI's with limited interactivity

2. Provider:
Pros:
- scoped state management:
provider allows localized and scoped statemanagement reducing the need for prop drilling

- Easy integration: It is easy to integrate into flutter applications and offers a good balance between simplicity and flexibility

- Large community support:
provider is widely used and has a good community support.

cons:
- Learning curve:
- Global scopes: In some cases global ~~scope~~ state might be unintentionally created.

suitable scenarios:
- Applications of varying sizes with moder-
-ate to complex UIs.

3. Riverpod!
Pros:
- scoped and flexible.
- provider inheritance.
- Immuatable and reactive.
Cons:
- Learning curve: takes time to learn things intially.
- some advanced features may not require for simpler applications.

q.4> a> Explain the process of integrating firebase with flutter applications:
→ 1. create a firebase project:
- Go to firebase console and create a new project
- follow the setup instruction.

2. Add firebase to flutter application
- In your flutter projet, add firebase SDK depedencies to the `.yaml files`.

3. Intialize firebase
- Import the firebase packages and initialize in the `main.dart' file.

Date
Page

4. Configure firebase services:
 - Depending on the service you want to use (authentication firestore) configure them by following specific setup instructions provided by firebase.

5. Use firebase services in app.
 - implement firebase service in your app code.

Benefits of using firebase;
 1. Real-time database.
 2. Authentication.
 3. Cloud function
 4. Cloud firestore.
 5. Firebase storage.
 6. hosting and analytics.
 7. Authentication state management.
 8. secure and scalable.
 9. Easy setup and integration.

b) Highlight the services commonly used in flutter developement (firebase) and provide brief overview of how data synchronization is achieved.

→ 1. Authentication: firebase authentication for user sign-In.
 2. fire store: A NO-SQL database for real time data synchronization.

Teacher's Sign.:

Date
Page

3. firebase cloud messaging : Push notifi-
-cations for engaging users.

* Data synchronization:
1. Listeners and streams:
Flutter developers can use stream based API's ~~for~~ to listen for changes in data whether its in firestore, the realtime database or other firestore services

2. Reactively updating UI's : flutters 'stream builder' widget is commonly used to reactively update UI components based on ~~the~~ changes in data streams. When data changes on server, the stream emits the data, triggering a rebuild of the associated UI.

3. Offline support:
firebase services provide built in offline support. Flutter app can work seamlessly offline and when connectivity is restored, changes made offline are automatically synchronized with server.