(MAD & PWA Lab)

– Pranathi N.
D15B Roll no. 45

ASSIGNMENT - 1

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

Ans- Key features and advantages of Flutter are :

- Single codebase for multiple platforms :
  - It allows you to write code once and deploy it on both IOS and Android platforms, saving time and effort in maintaining separate codebases.
- Hot Reload :
  - Hot Reload enables developers to see changes in the app instantly, speeding up the development process and making it easier to experiment with UI changes.
- Rich and customizable UI :
  - Flutter offers a rich set of pre-built widgets and allows for highly customizable UI designs, enabling developers to create visually appealing and interactive user interfaces.
- High performance :
  - Flutter apps are compiled directly to native ARM code, which makes them performant and fast, with no need for Javascript bridge.
- Access to native features :
  - Flutter provides plugins that allow developers to access native features and APIs of the underlying platform, ensuring that they can integrate platform-specific functionalities seamlessly.
- Strong community and support :
  - Flutter has a growing community of developers, which means there are plenty of resources, libraries and tools available for developers.
- Support for material design and cupertino :
  - Flutter comes with built in support for both Material design (Android) and Cupertino (iOS) design languages, ensuring that apps feel and look native on each platform.

- Open source :

• Flutter is an open source framework backed by Google, which means it's continuously evolving with contributions from the community and has strong backing from a major tech company.

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

Ans- Flutter represents a departure from traditional mobile app development by introducing a single codebase for multiple platforms, streamlining the process of building apps for both iOS and Android. Its widget-based UI development model offers a highly flexible and customizable approach to creating user interfaces, allowing developers to craft complex designs with ease.

One of Flutter's standout features is its performance. By compiling directly to native ARM code, Flutter apps achieve high performance without the overhead of a Javascript bridge, resulting in smooth animations and responsive interfaces.

Flutter provides access to native features and APIs through its extensive plugin system, allowing users to integrate platform specific functionality seamlessly.

The framework's strong community and ecosystem, coupled with its open source nature has contributed to its widespread adoption. Developers benefit from a wealth of resources, libraries, and tools as well as ongoing updates and improvements, driven by a dedicated community and the backing of a major tech company. These factors have made flutter a popular choice for mobile app development, especially projects requiring cross platform compatibility, native performance and rapid development cycles.

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

Ans- In Flutter, the widget tree is a hierarchical structure where each widget represents a part of the user interface, and the arrangement of these widgets forms a tree. Widgets can be simple, like buttons or text fields, or complex, representing entire screens or custom components. The tree starts with a top-level widget like MaterialApp or CupertinoApp, which contains other widgets as its children, and these children can have their own children, forming their own ~~children~~ tree like structure.

Composition is a key principle in Flutter's widget tree, emphasising the building of complex UIs by combining simpler widgets. Instead of relying on inheritance for customisation, Flutter encourages developers to compose widgets together to create more complex and customizable components. This compositional approach offers flexibility, reusability, and maintainability, as developers can create and customize widgets independently, then ~~custom~~ combine them to build more complex UIs. It also helps in breaking down the UI into managable parts, making the codebase easier to understand and maintain.

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

Ans- Consider an example of building a widget tree for a login screen, commonly used widgets are:

1. MaterialApp: This is the root widget of the application, providing a framework for the app's, providing a framework for the app's material design. It contains various configuration options for the app and serves as the entry point for the widget tree.

2. Scaffold : Inside the MaterialApp, a scaffold widget is often used to implement the basic material design visual layout structure. It provides a framework for the app's basic visual layout, including app bars, drawers, and a bottom navigation bar.

3. App Bar : Within the scaffold, an AppBar widget can be used to display a material design app bar at the top of the screen. This widget typically contains a title and optional actions such as icons or buttons.

4. Body : Inside the scaffold, the body property is used to specify the main content of the screen. Container widget could contain other widgets like text fields and buttons.

5. Column : Inside the container widget we can use a column widget to arrange multiple widgets vertically. This allows to stack widgets on top of each other in a vertical layout.

6. TextFormField : Within the column, TextFormField widget can be used to create a text input field for the user to enter their username or email.

7. Elevated Button : Also within the column, ElevatedButton widget to create a button for the user to submit the login form.

Q3. (a) Discuss the importance of state management in Flutter applications.

Ans- 1. Consistent UI :

- Effective state management ensures that the UI reflects the most up-to-date data, providing users with a consistent and reliable experience.

2. Performance :

- Proper state management ensures improved performances by minimizing unnecessary UI updates and reducing the risk of memory leaks, leading to a smoother user experience.

3. Code organization:

- Well organized state management separates concerns and improves the maintainability of the code base, making it easier to understand and modify the app's logic.

4. Scalability:

- As the app grows in complexity, good state management practices make it easier to scale the app and add new features without introducing bugs or unexpected behaviour.

5. Local state:

- Local scate management is essential for handling UI-related state within specific widgets or their sub trees, such as managing visibility or handling user input.

6. Global state:

- Managing global state is crucial for sharing data across different parts of the app, such as user authentication status, theme preferences, or fetched data from APIs.

(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.

Ans- 1. setState:

- setState is a method provided by the StatefulWidget class in Flutter. It is used to update the state of a widget and trigger a rebuild of UI.
- Suitable scenarios: setState is suitable for managing local state within a singe widget or its subtree. It is ideal for simple UI interactions or small apps where the state changes are limited to specific widgets.

2. Provider:

- Provider is a popular state management solution in flutter that uses

Inherited widget and change notifier to propogate changes in state down the widget tree.

- Suitable scenarios: Provider is suitable for managing both local and global state in Flutter applications. It is ideal for medium to large sized apps where the state needs to be shared across multiple widgets or when the app's state management requires a more structured approach. Provider is also suitable for cases where you want to use a lightweight and flexible state management solution.

3. RiverPod:

- RiverPod is an improvement over Provider and is based on the provider package. It introduces a more modern and powerful approach to dependency injection and statement in Flutter.
- Suitable scenario: Riverpod is suitable for complex applications with a large number of dependencies and complex state management requirements. It is also suitable for cases where you want to leverage the full power of dependencies injection and take advantage of features like asynchronous dependencies and lazy loading.

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

Ans- Steps to integrate firebase with flutter application:

1. Create a firebase project:
   - Go to the firebase console and create a new project and follow the setup intructions.
2. Add Firebase to Flutter project:
   - Add the 'firebase_core' and other Firebase plugins to your 'pubspec.yaml'

- Run 'flutter pub get' to install the packages.

3. Configure firebase in Flutter :

- Initial Firebase in your Flutter app by calling 'Firebase.initializeApp()' in the main() function.

4. Set up Firebase services :

- Depending on the services you want to use (Authentication, firestore, realtime database), add the corresponding firebase plugins to your 'pubspec.yaml' file.
- Follow the setup instructions for each service to configure in app.

5. Use Firebase services in Flutter app: ~~Import~~

- Once Firebase is set up, you can use its services in your Flutter app by importing the necessary packages and making API calls to Firebase.

Firebase offers several benefits as a backend solution for flutter apps:

1. Real time database: Firebase synchronises data across devices in real time, ideal for applications needing live updates.
2. Authentication : It provides secure user authentication with various methods like email / password, social media and anonymous auth.
3. Cloud firestore : Firestore is a flexible, scalable NoSQL cloud database that integrates seamlessly with Flutter.
4. Cloud functions : Firebase allows extending app functionality with serverless cloud functions triggered by firebase events.
5. Hosting and storage : Firebase offers hosting for web apps and file storage with Firebase storage.

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

Ans- 1. Firebase authentication: Provides secure user authentication using email /password, phone number, social media logins and more.

2. Cloud Firestore : A flexible, scalable NoSQL cloud database for storing data syncing app data across devices in real time.
3. Firebase realtime database : A cloud - hosted database that supports data synchronization among users in real time.
4. Firebase storage : A cloud storage service for storing and serving user-generated content like images and videos.
5. Firebase cloud messaging : Allows sending push notifications to users across platforms.

Firebase achieves data synchronisation through its real-time capabilities in cloud firestore and firebase realtime database. When data changes are immediately propogated to all connected clients in real time. This means that if one user makes a change to a piece of data, all other users subscribed to that data will see the change reflected in their app without needing to refresh. This real-time synchronization is achieved through the use of web-sockets, which provide a persistant connection between the client and the server, allowing for efficient and instantaneous data updates. This capability is especially useful for collaborative apps, chat apps, live data feeds, and any other scenarios where real-time updates are essential for the user experience.