Flutter Interview Questions and Answers

1. Basic Flutter Questions

1.1 What is Flutter, and why is it used?

Flutter is an open-source **UI toolkit** by Google for building **cross-platform apps** (iOS, Android, web, desktop) from a single **codebase**. We use it for its fast development, **native performance** via **AOT compilation**, and customizable **widgets** for great-looking screens.

1.2 Why is Flutter better than other frameworks?

It uses one **codebase** for all platforms, supports **hot reload** for instant UI updates, and provides a rich **widget library** for flexible designs with **native performance**.

1.3 What's a Widget?

A **widget** is a building block for the app's **UI**. Everything—like **Text**, **Button**, or **Container**—is a widget. They're either **StatelessWidget** (fixed) or **StatefulWidget** (dynamic).

1.4 What's the difference between StatelessWidget and StatefulWidget?

A **StatelessWidget** is immutable, like a static **Text** widget. A **StatefulWidget** manages **state** and rebuilds when data changes, like a **Counter** widget updating its value.

1.5 What's the `build` method?

The `build` method defines a widget's **UI** in the **widget tree**. It runs during **rendering** or when **state** updates, returning a tree of widgets to display.

2. Architecture and Core Concepts

2.1 What's the Widget Tree?

The **widget tree** is a hierarchy of **widgets** that defines the app's **UI**. Flutter uses it to create a **render tree** for displaying the interface on the screen.

2.2 What's the difference between `MaterialApp` and `WidgetsApp`?

`MaterialApp` provides a **Material Design** structure with **navigation**, **theming**, and **scaffolding**. `WidgetsApp` is a basic **app framework** without Material Design for custom UIs.

2.3 What's `BuildContext`?

`BuildContext` is an object that locates a widget in the **widget tree**. It helps access **inherited widgets**, like **ThemeData**, or manage **navigation**.

2.4 What's a `Scaffold`?

`Scaffold` is a **widget** that sets up a **Material Design** screen with components like **AppBar**, **Drawer**, or **FloatingActionButton** for quick UI setup.

2.5 What's the `pubspec.yaml` file?

The `pubspec.yaml` is a **configuration file** that lists **dependencies** (like packages), **assets** (images, fonts), and app metadata (name, version).

3. State Management

3.1 What's state management, and why is it important?

State management handles **state** (data) changes to update the **UI**. It's key for making apps **responsive** to user interactions or data updates.

3.2 What are some state management tools?

Basic: `setState`. Advanced: **Provider**, **Riverpod**, **Bloc**, **Redux**, **GetX**, and **MobX**.

3.3 How does `Provider` work?

Provider is a **state management** package that shares data via **dependency injection**. You wrap the app with a **Provider** widget, and children use **Consumer** to rebuild when data changes.

3.4 What's the difference between `setState` and Bloc?

`setState` rebuilds the **UI** locally but can clutter code. **Bloc** uses **streams** to separate **business logic** from the **UI**, making apps scalable and testable.

3.5 When do you use `InheritedWidget`?

Use `InheritedWidget` to share data (like **ThemeData** or settings) across the **widget tree** efficiently, avoiding manual prop passing.

4. UI and Layout

4.1 What's the difference between `Row` and `Column`?

A `Row` arranges **widgets** horizontally (side by side). A `Column` arranges them vertically (stacked). Both use **MainAxisAlignment** for positioning.

4.2 How do you make an app responsive?

I use `MediaQuery` for **screen size**, `LayoutBuilder` for widget constraints, and **Flexible/Expanded** widgets to adapt the **layout** to different devices.

4.3 What's `SafeArea`?

`SafeArea` is a **widget** that adds padding to avoid **system overlays** (like notches or status bars), keeping the **UI** visible.

4.4 How do you make a custom widget?

I create a class extending **StatelessWidget** or **StatefulWidget**, define properties (e.g., text), and return a **UI** in the `build` method, like a custom **ElevatedButton**.

4.5 What are `Keys`, and when are they used?

Keys identify **widgets** during **widget tree** rebuilds. I use **ValueKey** or **UniqueKey** in lists to preserve **state**, like tracking list items.

5. Networking and APIs

5.1 How do you fetch data from the internet?

I use the `http` package to make **HTTP requests**, like `http.get`, to fetch **JSON** data from an **API** and display it.

5.2 What's `Future` and `async/await`?

A `Future` represents a value available later, like an **API** response. `async/await` simplifies **asynchronous** code, making it read like normal code.

5.3 How do you handle errors in network calls?

I use `try-catch` with `async/await` to catch **exceptions** (e.g., no internet) and show a user-friendly **error message**.

6. Navigation

6.1 How do you switch screens?

I use `Navigator.push` to add a new **route** (screen) and `Navigator.pop` to go back, like flipping pages in a stack.

6.2 What's `push` vs. `pushReplacement`?

`Navigator.push` adds a new **route** to the **navigation stack**, allowing back navigation. `Navigator.pushReplacement` replaces the current **route**, preventing going back.

6.3 What's `Named Navigation`?

Named Navigation uses **route names** defined in `MaterialApp`'s `routes` map. I call `Navigator.pushNamed` to jump to a screen by name.

7. Performance and Optimization

7.1 How do you make an app fast?

I use `const` constructors for static **widgets**, `ListView.builder` for efficient lists, and **Flutter DevTools** to optimize **performance**.

7.2 What's `ListView` vs. `ListView.builder`?

`ListView` builds all **children** at once, good for small lists. `ListView.builder` uses **lazy loading** to build items as they appear, better for large lists.

7.3 What's a `const` constructor?

A `const` constructor creates a **compile-time constant** widget, reducing rebuilds and improving **performance**.

8. Testing

8.1 What tests are supported in Flutter?

Unit tests for functions, **widget tests** for **UI** components, and **integration tests** for app flows.

8.2 How do you write a unit test?

I use the `test` package to write a **unit test**, checking if code works, like verifying `1 + 1 == 2`.

8.3 What's `WidgetTester`?

`WidgetTester` simulates **user interactions** (taps, typing) in **widget tests** to verify **UI** behavior, like checking a button's text.

9. Advanced Topics

9.1 What's Dart in Flutter?

Dart is Flutter's programming language. It supports **JIT compilation** for **hot reload** and **AOT compilation** for fast apps.

9.2 What's Hot Reload?

Hot Reload updates the **UI** instantly when code changes, speeding up development without restarting the app.

9.3 What's an `Isolate`, and when is it used?

An `Isolate` is a separate **thread** for heavy tasks (e.g., complex calculations) to prevent **UI jank** in the main thread.

9.4 How do you add platform-specific features?

I use **platform channels** (`MethodChannel`) to call **native code** (Kotlin/Swift) for features like the camera.

10. Practical Questions

10.1 How do you add a dark theme?

I configure `ThemeData.light()` and `ThemeData.dark()` in `MaterialApp` and set `themeMode: ThemeMode.system` to switch based on device settings.

10.2 How do you support different languages?

I use the `intl` package, define translations in `.arb` files, and access them via `AppLocalizations` for **localization**.

10.3 How do you debug an app?

I use **Flutter DevTools** for **performance profiling**, the **widget inspector** for **UI** issues, and `debugPrint` or breakpoints for debugging.

10.4 How do you handle deep linking?

I use packages like `uni_links` to handle **deep links**, configure platform settings (e.g., Android intents), and route to specific screens.