FLUTTER - INTERVIEW SECRETS

General Flutter Developer Interview Questions

Which skills are required to use Flutter?

Knowledge of the Dart programming language
Knowledge of the Flutter framework
Android mobile development skills
iOS mobile development skills

Explain what Flutter is?

Flutter is an open-source framework and toolkit developers use to create applications using the Dart programming language.

Name four advantages of using Flutter?

Flutter enables effortless development across multiple platforms
There's detailed Flutter documentation available, which developers can refer to
It features a handy JIT feature for increasing the development speed and facilitating UI refreshes
Developers can have easy access to information from the Flutter community

Explain what Dart is?

Dart is an object-oriented programming language that uses a syntax similar to C. Dart use when creating an app using the Flutter framework.

Describe what a stateful widget is?

A stateful widget is a widget that user interactions can change. This type of widget is dynamic, meaning that its appearance can change when send data to it.

Such as slider, radio button, checkbox, etc.

Write the advantages of using flutter?

For developing mobile applications, Flutter usually fulfills the custom needs and requirements. It offers the following advantages:

Reduce Code Development: Flutter's hot reload feature allows it to offer faster performance. With it, the application gets compiled using the arm C/C++ library, making it closer to machine code and enabling it to run more quickly. The Flutter team has put lots of effort into providing a wide variety of ready-to-use widgets. Most of them are incredibly customizable, saving your time like no other framework before.

Cross-platform Development: Using Flutter, you can write code, manage, and run it across multiple platforms. For the developers, this saves time, money, and effort.

Live and Hot Reloading: This makes the app development process simpler and faster. Additionally, it also allows us to modify or update the code once a change is made.

Similar to Native App performance: In contrast to most cross-platform frameworks, Flutter does not rely on intermediate code representations or interpretations. The Flutter application is built directly into the machine code, which eliminates any performance issues associated with the interpretation process. With Flutter, you get a fully compiled release application ahead of time.

Good Community Support: Developers can ask questions about issues and get answers quickly.

Little/Minimal Code: Each Flutter app is built using Dart programming language, which uses JIT and AOT compilation for faster startup time, faster performance, and smoother functionality. With the JIT feature, you can increase the speed of development and refresh the UI.
Documentation: Flutter's documentation is well-organized and informative. It serves as a central repository for all written documents.
Expressive and Flexible UI: Flutter offers a customizable layered architecture that allows for highly customizable designs, expressive UIs, and fast rendering.

Describe what a stateless widget is?

A stateless widget is the opposite of a stateful widget in that user interactions cannot change it. Stateless widgets, including icons, image, and icon buttons, etc.

When would you use profile mode in Flutter?

Developers use profile mode to maintain debugging abilities and analyze an app’s performance while it’s tested. Compiling profile mode requires the flutter run –profile command.

When would you use release mode in Flutter?

Developers use release mode in Flutter to deploy an app, reduce the footprint size, and increase optimization. Applicants should know that release mode features include a faster startup and execution. Developers can compile release mode using flutter run –release.

When should you use keys in Flutter?

Keys are best used when developers want to preserve a modified widget’s state? Developers can use keys to reorganize widget connections and widget trees, particularly when the widget trees contain stateful widgets.

What is the difference between an Isolate and a Future?

Dart, despite being a single-threaded language, offers support for multithreading. Future class, often combined with async/await sugar code, allows us to create asynchronous code. However, this code will be executed on a single thread. If we need to perform a non-trivial operation that would benefit from using multiple threads we can use Isolate. By calling the compute method or Isolate.spawn constructor, we can create an isolate that will work independently to the UI. Using isolates can help if our animations are dropping framerate while executing demanding code.

What is your favorite state management?

As Senior Developers, we should be able to operate on more than one popular state management solution. On our path through many different Flutter projects, we can familiarize ourselves with packages like bloc, riverpod, provider, or hooks. Some engineers, especially the ones that come from web software development, like to adapt Redux or MobX into their projects. The most popular solution for many developers seems to be a combination of different state management solutions. That's because the state in Flutter can be divided between ephemeral state and app state.

How would you approach implementing a paginated list view in Flutter?

There are two approaches to solve this problem. One is to use the infinite_scroll_pagination package. This approach allows us to implement simple paginated list views quickly. A lot of the boilerplate code is taken care of for us. However, on rare occasions, the requirements disqualify using a premade solution like infinite_scroll_pagination. Custom UI design or an uncommon way of calculating pages and list items may require us to build the pagination using widgets that exist in Flutter. In that case, a good knowledge of ListView and GridView may come in handy.

What is a WidgetsBindingObserver?

Flutter apps work in a lifecycle. That means that an app can have many states. For example when a user turns off the screen by blocking their phone the state of the app is changed. WidgetsBindingObserver is a mixin that allows reading changes in app state. States that can occur are inactive, paused, resumed, and suspending.

Based on that information we can perform operations like stopping an audio player after the screen is turned off.

What is your favorite dependency injection solution and why? Why is it so important to do proper dependency injection in our Flutter projects?

In Flutter there are two approaches to dependency injection. One assumes that dependency injection should be performed in the widget tree. This approach can be achieved by using the InheritedWidget or the provider package. The other one assumes that the dependency injection is performed outside of the widget tree. We can perform this by using the get_it package. This package recognizes itself as a Service Locator. Some developers view this approach as a non-Flutter approach. However, it was proven by many production projects that it's a viable solution for dependency injection in Flutter apps. In combination with the injectable package, this can be a perfect match for managing many dependencies in a large-scale project.

Dependency injection is important to manage the objects registered in the memory of our device. We can make sure that objects that are supposed to be registered in memory once are registered properly and not recreated within the code's logic. Also, proper dependency injection allows us to create testable code.

What local storage solutions did you use? When is it more advantageous to use SQL rather than NoSQL solutions?

There are several approaches to save local data while programming in Flutter.

The first is to use the shared_preferences package. This approach allows you to save simple key-value pairs. It's perfect for saving simple, unorganized data.

A solution that allows us to save more complex data would be a NoSQL database. The most common package for that approach is the hive. It's a very fast database providing a simple, powerful, and intuitive API. It has built-in encryption using AES-256. What may be important for long-lasting projects is that it does not use any native dependencies. That means the chances of it breaking down during Flutter or OS update are very low.

The most complex solution would be to use SQL databases. That type of database allows you to use SQL queries to read and manipulate data. For complex solutions and apps that process a lot of structured and related data, this may be a perfect fit. The most popular solution for Flutter would be sqflite package. It's an SQLite wrapper for Android and iOS, as well as macOS. It supports transactions, batches, and automatic version management. If you don't want to write SQL there are helper methods for CRUD operations.

What is KISS principle design pattern?

Keep it simple, stupid (KISS) is a design principle which states that designs and/or systems should be as simple as possible. Wherever possible, complexity should be avoided in a system—as simplicity guarantees the greatest levels of user acceptance and interaction.

What is DRY pattern?

DRY, which stands for 'don't repeat yourself,' is a principle of software development that aims at reducing the repetition of patterns and code duplication in favor of abstractions and avoiding redundancy.

Are you familiar with concepts like Clean Architecture and Domain-Driven Design?

Clean Architecture and Domain-Driven Design are strategic approaches towards building complex enterprise apps. As a candidate for senior-level, you should be able to build apps based on at least one of these approaches. That should allow you to build scalable and testable applications.

Clean Architecture focuses on testability and modularity. The main business logic of our app should be separated from the UI and the rest of the code.

DDD is meant to establish a common language between developers and business experts.

Follow the KISS and DRY pattern

https://www.interviewbit.com/flutter-interview-questions

-