|  |  |
| --- | --- |
| Candidate: | **Rakesh Gurudu** |
| Role: | **Lead/Senior Android Developer** |

# 

|  |
| --- |
| 1. **Which is your favourite architectural pattern while developing android apps, Explain why?**   **MVVM:** |
| MVVM stands for Model, View and ViewModel. Model is made up of the data and business logic. View handles the UI and ViewModel helps in communication between View and Model via LiveData. This creates separation and reduces tight coupling on each other. The View related classes. The components depends on only the components bbelow it for eg. View->ViewModel->Model. This separation also helps in testing each component independently.  The Model is usually implemented using a Repository class which gets the data from network or locally based on the business logic and conditions.  LiveData is used for observing data changes. LiveData is lifecycle aware which has an advantage in which it gets cleared when the activity or fragment is destroyed. ViewModel also handles activity configuration changes automatically. |
| 1. **Explain Kotlin coroutines with a sample use for it** |
| While doing asynchronous function calls we can using callback to get the result. But nested callbacks becomes hard to track and read. Kotlin coroutines are light weight thread used for async calls. Functions are marked with suspend which do not block the caller thread. Coroutines easily switches the thread using dispatchers. Coroutine cancels other coroutines it creates when the parent coroutine is cancelled using scopes. When we call a suspend function from launch it waits till the suspend function completes to execute the next line written in launch.  For eg. Consider we want to get to movie list by making a network call to server. By using callbacks we can write a function as shown below:  fun getMovieList(){  fetchMoviesFromServer{  movieList -> {  updateUI(movieList)  }  }  }  The same with coroutines is simpler to read and easy to understand:  suspend fun getMovieList(){  val movieList = fetchMoviesFromServer()  updateUI(movieList)  } |
| 1. **Explain solid principles that you have applied to your projects with example** |
| With MVVM pattern every class a single responsibility for eg fragment class handles the UI, ViewModel class handles the business logic and data.  Using dagger dependency injection is achieved which helps in separating modules based on functionality and better testing. |
| 1. **What are BuildTypes and product flavours in Gradle? And What can you use them for?** |
| Product flavour is used to provide different functionality of the same app for eg. an app can have a free version with basic functionality and a paid version with modules which provide extra functionality. It can also be used if a vendor wants to provide same same app to different clients with some modified code based on the individual client requirement.  Build types do not change functionality for eg. an app can have a release build which has code obfuscated with progaurd, the files compressed to reduce app size, live keys for third-party sdk etc. whereas with debug build we can have logging for api’s and other logs turned on for debugging. The main user functionality would be same in both builds. In debug we can also turn off extra optimizations to improve build times during development, ability to use prod and pre-prod api’s in same debug build to perform testing |
| 1. **You Just launched your application to play store that supports Inapp purchases for unlocking features in the app.some of your end users are complaining via Appstore reviews that payments are being deducted but the features are yet to be unblocked.What details will you need to analyse this issue** |
| The account details of the user and order id or transaction id which is received to the user from payment gateway via email when payment deduct |