4. Command line tools

4.1 apkanalyzer – Analyse Build Files

The APK analyser is available after the app has been build and provides the opportunity to inspect the resources of the app. Once the App has been built the final (file) versions that are in included in the app, such as the file's contents, size and more, can be analysed with this tool.

4.2 adb – Controlling / Communication with devices

This tool is used to communicate with devices. When this command is executed the first, time and thus no server is running, then the a server will be started that automatically connects to all running devices (emulators included). When the server has finished its setup, then the client can send commands to the devices using further adb commands.

4.3 lint – Detection of imperfect code

Lint is a tool that checks the code of the app for structural problems, thus code fragments that may impact the apps performance (due to lack of efficiency) or its reliability. Those problems are then displayed with a message and a severity level.

5. Gradle

5.1 Term definitions (build type, product flavour, build variants, dependencies, multi APKs)

Gradle is used for 'building' projects. Each project or group of projects has to have a *build.gradle* file, which includes the dependencies, plugins, configurations of the project /s as well as tasks. Typical tasks are the compilation of the code, running the projects tests and more.

Build type

The build type is part of the *build.gradle* file and sets properties for different stages. Thus, when the project is in development, then the project may work with different build settings that it would when it is released. By default the build file has two types debug and release, and their main difference that the debug type allows the developers to debug the project.

Product flavour

This one is another property inside the *build.gradle* file and is used when there are different versions of a project, which have different device and resource requirements. Thus, different "*flavours*" generally speaking have different requirements.

Build variants

Each *build variant* may represent a different app on the Play Store, like pro and free version, of the same project, but with different features and resources.

Dependencies

When working on bigger projects one will rely on already existing packages, such as Socket.io, which come with code that will be embedded and therefore become accessible inside the project. By declaring *dependencies* and thus importing them, one can use existing and thoroughly tested code fragments.

Multi APKs

This term is used when an app has to support all or most target devices and thus has to support multiple screen resolutions and internal data structures (Application Binary Interfaces). In order to, reduce a single APK file into multiple smaller ones, where each of those APK files are used for their specific range of devices, helps to reduce the overall size of each APK file.

5.2 Build process description

Compiling: Firstly, all the code files, of the project, (kotlin/java) are converted to dalvik bytecode.

Resource Processing: Then the apps resources, such as the layout files, images or (environment) strings are processed.

Packaging: Once the code and resources have been processed, both of those information's are packaged as a an APK file. (Packaging)

Signing: After the APK file has been created, a digital signature is being added which states the integrity of its content.

Installation: Finally the signed APK file is installed on the Android device or made available to users on the Play Store or other channels.

5.3 Gradle files (settings.gradle, build.gradle, gradle.properties)

5.3.1 build.gradle

As previously mentioned, the build.gradle file includes the projects dependencies, plugins, … and tasks for its compilation.

5.3.2 settings.gradle

In contrast to the build.gradle file the settings.gradle file is executed first. With it subprojects can be added to the build, the parameters of the command line can be modified or it also has access to the Gradle object itself.

5.3.3 gradle.properties

This file stores keys and their values inside of it. Since it stores values that may be used to override existing settings.