1. **IMPLEMENTATION PLANNING**
   1. **IMPLEMENTATION ENVIRONMENT**

Android Studio is the official [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) for [Google](https://en.wikipedia.org/wiki/Google)'s [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) [operating system](https://en.wikipedia.org/wiki/Operating_system), built on [JetBrains](https://en.wikipedia.org/wiki/JetBrains)' [IntelliJ IDEA](https://en.wikipedia.org/wiki/IntelliJ_IDEA) software and designed specifically for [Android development](https://en.wikipedia.org/wiki/Android_software_development). It is available for download on [Windows](https://en.wikipedia.org/wiki/Windows), [macOS](https://en.wikipedia.org/wiki/MacOS) and [Linux](https://en.wikipedia.org/wiki/Linux) based operating systems It is a replacement for the [Eclipse Android Development Tools](https://en.wikipedia.org/wiki/Eclipse_(software)#Android_Development_Tools) (ADT) as primary IDE for native Android application development.

Android Studio was announced on May 16, 2013 at the [Google I/O](https://en.wikipedia.org/wiki/Google_I/O) conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0. The current stable version is 3.2.1, which was released in October 2018

The following features are provided in the current stable version:

* [Gradle](https://en.wikipedia.org/wiki/Gradle)-based build support
* Android-specific [refactoring](https://en.wikipedia.org/wiki/Code_refactoring) and quick fixes
* [Lint](https://en.wikipedia.org/wiki/Lint_(software)) tools to catch performance, usability, version compatibility and other problems
* [ProGuard](https://en.wikipedia.org/wiki/ProGuard_(software)) integration and app-signing capabilities
* Template-based wizards to create common Android designs and components
* A rich [layout editor](https://en.wikipedia.org/wiki/Graphical_user_interface_builder) that allows users to drag-and-drop UI components, option to [preview layouts](https://en.wikipedia.org/wiki/WYSIWYG) on multiple screen configurations[[16]](https://en.wikipedia.org/wiki/Android_Studio#cite_note-16)
* Support for building [Android Wear](https://en.wikipedia.org/wiki/Android_Wear) apps
* Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud Messaging (Earlier 'Google Cloud Messaging') and Google App Engine[[17]](https://en.wikipedia.org/wiki/Android_Studio#cite_note-17)
* Android Virtual Device (Emulator) to run and debug apps in the Android studio.
  1. **Program/Modules Specification**

Qr code module: Scans qr code.

Firebase module: saves data at backend.

* 1. **Security Features**

Authentication is secured by firebase authentication.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

Authenticate users with their email addresses and passwords. The Firebase Authentication SDK provides methods to create and manage users that use their email addresses and passwords to sign in. Firebase Authentication also handles sending password reset emails.

Attendance is taken securely scanning a randomly generated qr code which constantly changes after 10 seconds so no one can have proxy attendance or fraud attendance.

* 1. **Coding Standards**

Proper and consistent indentation is important in producing easy to read and maintainable programs. Indentation should be used to:

• Emphasize the body of a control statement such as a loop or a select statement

• Emphasize the body of a conditional statement

• Emphasize a new scope block

A minimum of 3 spaces shall be used to indent. Generally, indenting by three or four spaces is considered to be adequate. Once the programmer chooses the number of spaces to indent by, then it is important that this indentation amount be consistently applied throughout the program. Tabs shall not be used for indentation purposes.

Examples:

/\* Indentation used in a loop construct. Four spaces are used for indentation. \*/

for ( int i = 0 ; i < number\_of\_employees ; ++i )

{

total\_wages += employee [ i ] . wages ;

}

// Indentation used in the body of a method.

package void get\_vehicle\_info ( )

{

System.out.println ( “VIN: “ + vin ) ;

System.out.println ( “Make: “ + make ) ;

System.out.println ( “Model: “ + model ) ;

System.out.println ( “Year: “ + year ) ;

}

/\* Indentation used in a conditional statement. \*/

IF ( IOS .NE. 0 )

WRITE ( \* , 10 ) IOS

ENDIF

10 FORMAT ( “Error opening log file: “, I4 )

Inline Comments :

Inline comments explaining the functioning of the subroutine or key aspects of the algorithm shall be frequently used. See section 4.0 for guidance on the usage of inline comments.

Structured Programming :

Structured (or modular) programming techniques shall be used. GO TO statements shall not be used as they lead to “spaghetti” code, which is hard to read and maintain, except as outlined in the FORTRAN Standards and Guidelines.

Classes, Subroutines, Functions, and Methods :

Keep subroutines, functions, and methods reasonably sized. This depends upon the language being used. For guidance on how large to make software modules and methods, see section 4.0. A good rule of thumb for module length is to constrain each module to one function or action (i.e. each module should only do one “thing”). If a module grows too large, it is usually because the programmer is trying to accomplish too many actions at one time.

The names of the classes, subroutines, functions, and methods shall have verbs in them. That is the names shall specify an action, e.g. “get\_name”, “compute\_temperature”.

Source Files :

The name of the source file or script shall represent its function. All of the routines in a file shall have a common purpose.

Variable Names :

Variable shall have mnemonic or meaningful names that convey to a casual observer, the intent of its use. Variables shall be initialized prior to its first use.

Example:

The variable names should be in camel case letters starting with a lower case letter. For example, use 'total Amount' instead of 'Total Amount'.

Use of Braces :

In some languages, braces are used to delimit the bodies of conditional statements, control constructs, and blocks of scope. Programmers shall use either of the following bracing styles:

for (int j = 0 ; j < max\_iterations ; ++j)

{

/\* Some work is done here. \*/

}

or the Kernighan and Ritchie style:

for ( int j = 0 ; j < max\_iterations ; ++j ) {

}

It is felt that the former brace style is more readable and leads to neater-looking code than the latter style, but either use is acceptable. Whichever style is used, be sure to be consistent throughout the code. When editing code written by another author, adopt the style of bracing used.

Braces shall be used even when there is only one statement in the control block. For example:

Bad:

if (j == 0)

printf (“j is zero.\n”);

Better:

if (j == 0)

{

printf (“j is zero.\n”);

}

Summary of Guidelines followed and not followed by us during our Project Coding Implementation.

|  |  |
| --- | --- |
| Name of Coding Standard | Followed(Y)/Not Followed(N) |
| Inline Comments | N |
| Structured Programming | Y |
| Classes, Subroutines, Functions, and Methods | Y |
| Source Files | Y |
| Variable Names | Y |
| Use of Braces | N |