**CHAPTER-1**

**INTRODUCTION**

Integrated development environments are designed to maximize programmer productivity by providing tight-knit components. IDEs present a single program in which all development is done. This program typically provides many features for authoring, modifying, compiling, deploying and debugging software. Developing application for next-generation mobile phones using the open source android SDK is quite interesting. Application like games maybe their lab for various android devices like phones, tablets. In a Java language environment, android provides a rich application framework that allows us to build innovative applications for mobile devices. In android studio, there is flexibility of Gradle-based build system and there are multiple build variants and APK file generation.  Android studio uses gradle as the foundation of the build system, with more android-specific capabilities provided by the android plugin for gradle.

The application developed is called Quantico, Which is an information and education based application. This application can be used to retrieve information from the Internet and also manipulate the retrieved Data using interfered user's task related data. Every day, in the life of a normal person it is necessary to have basic needs that satisfies his desires. This application fulfils the needs of a software or IT related entity in everyday life. This application is the one that can be downloaded from the android store. It is a very reliable app.

This application involves the use of mathematical operations information retrieval and user's task. Quantico is an app that converts the amount of money from one currency to another currency. For this to happen there has to be a currency conversion factor for a constraint. This constraint is retrieved from the Internet and the conversion is made. This application has two drop down boxes that help the user to choose from the list of currencies and the converted value is displayed at real time. The time at which, the currency value has been updated is also displayed at the bottom of the screen at all times.

**1.1 PURPOSE OF THE PROJECT**

Quantico is used for the conversion of currency of one country into another currency. It saves a lot of time and effort by making everything in the app user friendly.  The available assets in the application are the:

* From: Currency1
* To: currency2
* Currency1: amount
* Currency2: result

It also has multiple options to select from. This application allows one to update the values either manually or automatically over Wi-Fi. In the automatic update option, there are multiple intervals for a news of the choose from. And also, and user can set the default 'from' and 'to' currencies.

**1.2 SCOPE OF PROJECT**

This project helps a user to find or know the value of a currency of one country in the equivalent terms of another country's currency. This is done at real time. It is very useful because it is an application that is compatible and there is no need to calculate the value of a currency. This had to use it to save and effectively use resources.

**CHAPTER-2**

**SOFTWARE ARCHITECTURE**

The design of the software is built upon several factors that are responsible for proper and effective use of the system resources in the device. The protocols and the methods followed during the operation are the foundation of the processing.

**2.1 INTRODUCTION**

The past decade has seen the emergence of mobile applications and the domination of smart phones offering variety of features through various applications. Mobile apps were originally offers for general productivity and information retrieval, including email, calendar, contacts, and stock market and weather information. However, public demand and the availability of developer tools drove rapid expansion into other categories, such as mobile games, factory automation, GPS and location-based services, banking, order-tracking, and ticket purchases. With the advancement in technology also arises the need to address the needs of the mobile users to a greater extent. Developers worldwide are competing to get better apps in every category with every little enhancement possible for progress. Moreover the numbers of Smartphone users are increasing exponentially and regular mobile devices are occasionally seen.

A mobile application (or mobile app) is a software application designed to run on smart phones, tablet computers and other mobile devices. They are usually available through application distribution platforms, which are typically operated by the owner of the mobile operating system, such as the Apple App Store, Google Play, Windows Phone Store, and BlackBerry App World. Some apps are free, while others must be bought. Usually, they are downloaded from the platform to a target device, such as an iPhone, BlackBerry, Android phone or Windows phone, but sometime they can be downloaded to laptops or desktops. For apps with a price, generally a percentage, 20-30%, goes to the distribution provider (such as iTunes), and the rest goes to the producer of the app. The term "app" has become popular, and in 2010 was listed as "Word of the year" by the American Dialect society. In 2009, technology columnist David Pogue said that newer smart phones could be nicknamed "app phones" to distinguish them from earlier less-sophisticated smart phones.

Mobile application development is the process by which application software is developed for low-power handheld devices, such as personal digital assistants, enterprise digital assistants or mobile phone. These applications can be preinstalled on phones during manufacturing, downloaded by customers from various mobile software distribution platforms, or client-side processing (e.g. JavaScript) to provide an "application-like" experience within a Web browser. Application software developers also have to consider a lengthy array of sizes, hardware specifications and configurations because of intense competition in mobile software and changes within each of the platforms. Mobile app development has been steadily growing, both in terms of revenues and jobs created. A 2013 analyst report estimates there are 529,000 direct App Economy jobs within the Au 28 members, 60% of which are mobile app developers. Each of the platforms for mobile applications also has an integrated development environment, which provides tools to allow a developer to write test and deploy applications into the target platform environment. Developing application software for mobile devices requires considering the constraint of these devices. Mobile devices run on battery and have less powerful array of screen sizes, hardware specifications and configurations because of intense competition in mobile software and changes within each of the platforms.

Mobile application development requires use of specialized integrated development environments made. Mobile applications are first tested within the development environment using emulators and later subjected to field testing. Emulators provide an inexpensive way to test applications on mobile phones to which developers may not have physical access.

In this category the Android operating systems heads the popularity list. It has emerged as a new mobile development platform, building on past successes while avoiding past failure of other platforms. Designed to empower mobile software developers to write innovative mobile applications, it is open source platform, with no up-front fees, and developers enjoy many benefits over other competing platforms. Touted as an innovative and open platform, Android has thus been positioned to address the growing needs of the mobile marketplace.

Android's share of the global Smartphone market, led by Samsung products, was 64% in March 2013. In July 2013 there were 11,868 models of Android devices, scores of screen sizes and eight OS versions simultaneously in use. The operating system's success has made it a target for patent litigation as part of the so-called "Smartphone wars" between technology companies. As of September 3, 2013, 1 billion Android devices have been activated.

Android's user interface is based on direct manipulation, using touch inputs that loosely correspond to real-world actions, like swiping, tapping and reverse pinching to manipulate on-screen objects. The response to user input is designed to be immediate and provides a fluid touch interface, often using the vibration capabilities of the device to provide hepatic feedback to the use. Internal hardware such as accelerometers, gyroscopes and proximity sensors are used by some applications to respond to additional user actions, for example adjusting the screen from portrait to landscape depending on how the device is oriented, or allowing the user to steer a vehicle in a racing game by rotating the device, simulating of a steering wheel.

Android devices bottom the home screen, the primary navigation and information point on the device, which is similar to the desktop found on PCs. Android home screens are typically made up of app icons and widgets; app icons launch the associated app, whereas widgets display live, auto-updating content such as the weather forecast, the user's email inbox, or a news tracker directly on the home screen. A home screen may be made up of several pages that the user can swipe back and forth between, though Android's home screen interface is heavily customizable, allowing the user to adjust the look and feel of the device to their tastes. Third party apps available on Google Play and other app stores can extensively re-theme the home screen, and even mimic the look of other operating systems, such as Windows Phone. Most manufacturers, and some wireless carriers, customize the look and feel of their Android devices to differentiate themselves from their competitors.

Present along the top of the screen is a status bar, showing information about the device and its connectivity. this status bar can be "pulled" down to reveal a notification screen where apps display important information or updates, such as a newly received email or SMS text, in a way that does not immediately interrupt or inconvenience the use. In earlier versions of Android these notifications could be tapped on to open the relevant app, but recent updates have provided enhanced functionality, such as ability to call a number back directly from the missed call notification without having to open the dialer app first. Notifications are persistent until read or dismissed by the user.

Android has a growing selection of third party applications. Which can be acquired by user’s either though an app store such as Google Play or the Amazon App Store, or by downloading and installing the application's APK file from a third-party site. The Play Store application allows user to browse, download and update apps published by Google and third-party developers, and is pre-installed on devices that comply with Google's compatibility requirements. The app filters the list of available application to those that are compatible with the user's device, and developers may restrict their applications to particular carriers or countries for business reasons. Purchases of unwanted applications can be refunded within 15 minutes of the time of download, and some carriers offer direct carrier billing for Google Play application purchases, where the cost of the application is added to the user's monthly bill.

Applications are developed in the Java language using the Android software development Kit (SDK). The SDK includes a comprehensive set of development tools, including a debugger, software libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) plug-in. Other development tools are available, including a Native Development Kit for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers, and various cross platform mobile web applications frameworks.

**2.2 DESCRIPTION OF TOOLS**

**2.2.1 Java**

Java is an object-oriented language, and is very similar to C++. Java is simplified to eliminate language features that cause common programming errors. Java source code files are compiled into a format called byte code, which can then be executed by a Java interpreter. Features being,

∙**Platform Independent:**The programs written on one platform can run on any platform provided the platform must have the JVM.

∙**Portable:**The feature Write-once-run-anywhere makes the java language portable provided that the system must have interpreter for the JVM.

∙**Simple:**Programs are easy to write and debug because java does not use the pointers explicitly. It also has the automatic memory allocation and de-allocation system.

∙**Multithreaded:**Multithreading means a single program having different threads executing independently at the same time.

∙**Robust:**Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other programming languages.

∙**Object Oriented:**To be an Object Oriented language, any language must follow at least the four characteristics.

i.Inheritance

ii.Encapsulation

iii.Polymorphism

iv.Dynamic binding

∙**Distributed:**The widely used protocols like HTTP and FTP are developed in java. Internet programmers can call functions on these protocols and can get access to the files from any remote machine on the internet rather than writing codes on their local system.

∙**Secure:**All the programs in java are run under an area known as the sand box. Security manager determines the accessibility options of a class like reading and writing a file to the local disk.

∙**High Performance:**In the beginning interpretation of byte code resulted in slow performance but the advance version of JVM uses the adaptive and just in time compilation technique that improves the performance.

∙**Integrated:**Java is an interpreted language as well. Programs run directly from the source code.

**2.2.2 Android Architecture**

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.

**Android Runtime**

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android.

**Application Framework**

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

**2.2.3 Android**

**What is Android?**

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

**Features:**

∙Application framework enabling reuse and replacement of components.

∙DALVIK virtual machine optimized for mobile devices.

∙Integrated browser based on the open source Web Kit engine.

∙Optimized graphics powered by a custom 2D graphics library.

∙3D graphics based on the OpenGL ES 1.0 specification.

∙SQLite for structured data storage.

∙Media support for common audio, video, and still image formats (JPG, MP3, etc).

∙GSM Telephony (hardware dependent).

∙Bluetooth, EDGE, 3G, and Wi-Fi (hardware dependent)

∙Camera, GPS, compass, and accelerometer (hardware dependent).

∙Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plug-in for the Eclipse IDE.

**Android Application**

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

**Libraries**

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed as follows:

**i.System C library**- a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices

**ii.Media Libraries**- based on Packet Video’s Open CORE; the libraries supportplayback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG

**iii.Surface Manager**- manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications

**iv.LibWebCore**- a modern web browser engine which powers both the Android browser and an embeddable web view

**v.3D libraries**- an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer

**vi. Free Type**- bitmap and vector font rendering

**vii.SQLite**- a powerful and lightweight relational database engine available to all applications

**Android Runtime**

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine.

Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management. An Android code editor that helps you write valid XML for your Android manifest and resource files. It will even export your project into a signed APK, which can be distributed to users.

To begin developing Android applications in the Eclipse IDE with ADT, you first need to download the Eclipse IDE and then download and install the ADT plug-in.

Following tools are required for the implementation:

* Java Programming
* XML Programming
* Awareness of the Android Development Tools
* Using Android Studio IDE

**Java Programming**

Java is a general purpose, concurrent, class based, object oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers “write once, run anywhere” (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to byte code(class file) that can run on any Java virtual machine(JVM) regardless of computer architecture .Java is, as of 2014,one of the most popular programming language in use, particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems’s Java platform. The language derives much of its syntax from C and C++, but has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1991 and first released in 1995.As of May 2007, in compliance with the specifications of the Java Community. Process, Sun relicensed most of its Java technologies under the GNU General Public License.

For the implementation of the proposed system, the developer is required to have knowledge regarding the following aspects in Java programming:

* **Structures and Syntaxes**

The developer should have a strong grip on all the structures and syntaxes used in Java as all the functional implementation of android application is done through the java file. All features present in Java can be used here where and when required.

* **Packages**

A Java package is a mechanism for organizing Java classes into namespaces similar to the modules of Modula. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

* A package provides a unique namespace for the types it contains.
* Classes in the same package can access each other’s package-access members.
* **Inheritance**

Inheritance can be the process where one object acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order.

When we talk about inheritance, the most commonly used keyword would be extends and implements. These words would determine whether on object acquire the properties of another object. Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.

The developer is expected to know the concept of inheritance mainly because the main class where the core work is carried out extends and/or implements various classes and hence he should have clear understanding of the working and mechanism.

**Xml Programming**

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is defined in the XML 1.0 specification produced by the W3C, and several other related specifications, all free open standards.

The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for the languages of the world. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

Many application programming interfaces (APIs) have been developed to aid software developers with processing XML data, and several schema systems exist to aid in the definition of XML-based languages.

The developer should have knowledge in the following features in XML:

* **Tag**

A mark-up construct that begins with < and ends with >. Tags come in three flavours:

* Start-tags; for example: <section>
* End-tags; for example: </section>
* Empty-element tags; for example: <line-break/>
* **Element**

A logical document component which earlier begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag. The characters between the start- and end-tags, if any, are the element’s content, and may contain markup, including other elements, which are called child elements.

* **Attribute**

A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag. **XML declaration**

XML documents may begin by declaring some information about them, as in the following example:

<? xml version=”1.0” encoding=”UTF-8”?>

These are important because Android programming would contain two important xml files, one for setting the layout, other for declaring permissions needed in the application.

**Android Development Tools**

Android Development Tools (ADT) is a plug-in for the Eclipse IDE that is designed to give you a powerful, integrated environment in which to build Android applications.

ADT extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, add packages based on the Android Framework API, debug your applications using the Android SDK tools, and even export signed (or unsigned) .apk files in order to distribute your application.

Developing in Eclipse with ADT is highly recommended and is the fastest way to get started. With the guided project setup it provides, as well as tools integration, custom XML editors, and debug output pane, ADT gives you an incredible boost in developing Android applications.

The Android SDK consists of android package which contains many classes that are needed for programming the application. The Android software development kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.10 or later, Windows XP or later; for the moment one can develop Android software on Android itself by using [AIDE- Android IDE- Java, C++] app and [Android java editor] app.

The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) Plug-in. Though IntelliJ IDEAIDE (All editions) fully supports Android development out of box, and Net Beans IDE also supports Android development via a plug-in. Additionally, developers may use any text editor to edit Java and XML files, then use command line tools (Java Development Kit and Apache ant are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).

Enhancements to Android’s SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

Android applications are packaged in .apk format and stored under /data/app folder on the Android OS (the folder is accessible only to the root user for security reasons). APK package contains .dex files (compiled byte code files called Dalvik executables), resource files, etc.

In addition to Eclipse’s standard editor features, ADT provides custom XML editors to help you create and edit Android manifests, resources, menus, and layouts in a form based or graphical mode. Double-clicking on an XML file in Eclipse’s package explorer opens the appropriate XML editor.

**Android Studio IDE**

Android Studio is essentially the community edition of JetBrains' IntelliJ IDEA with an Android plugin; a setup that will sound familiar to anyone who's used Eclipse with ADT. Previously, Eclipse has been Google's recommended IDE for Android, but now the search giant is shaking things up by giving developers a choice of supported development environments. But how does Android Studio stack up against Eclipse?

Firstly, Android Studio is currently only available as an early access preview, so you should expect missing functionality and perhaps even the odd bug, whereas Eclipse is an established and long-standing IDE. However, Android Studio is built on the proven IntelliJ platform and has the weight of Google behind it, so it's no surprise to find that it's very reliable and stable.

Another issue that all new software must address when it's trying to replace an established solution is users who are interested in trying something new, but have ongoing work tied to another solution. The early access preview handles this with wizards dedicated to migrating projects created within Eclipse, to Android Studio. This ensures that a half-completed Eclipse project is no barrier to trying out the new IDE. As an added bonus, once users have migrated they'll find many familiar tools waiting for them, such as the SDK and AVD managers.

In terms of new functionality, Android Studio offers template-based wizards for some of the most common Android designs and components, making it easier for developers to achieve a standard look and feel. Developers can also use the 'Gradle' build system that's included in Android Studio to automate building, testing, publishing and deployment.

One of the most exciting features of Android Studio is the innovative live layout editing mode that lets you preview an app's user interface across a range of devices. You can even preview an app across multiple devices simultaneously using the 'Preview All Screen Sizes' option, or toggle between different orientations, themes and platform versions. This functionality could prove to be a real time-saver for developers wrestling with the tricky task of designing for multiple screens.

**Developing in Android Studio with SDK Manager**

The Software Development Kit(SDK) plug-in for Android Studio adds powerful extensions to the Android Studio integrated development environment. It allows you to create and debug Android applications easier and faster. If you use the Studio, the SDK plug-in gives you an incredible boost in developing Android applications:

It gives you access to other Android development tools from inside the Studio IDE. For example, SDK lets you access the many capabilities of the DDMS tool: take screenshots, manage port-forwarding, set breakpoints, and view thread and process information directly from Eclipse.

It provides a New Project Wizard, which helps you quickly create and set up all of the basic files you'll need for a new Android application. It automates and simplifies the process of building your Android application.

The Android SDK includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator (based on QEMU), documentation, sample code, and tutorials. Currently supported development platforms includex86-architecture computers running Linux (any modern desktop Linux distribution), Mac OS X 10.4.8 or later, Windows XP or Vista. The officially supported integrated development environment (IDE) is Eclipse (3.2 or later) using the Android Development Tools (ADT) Plug in, though developers may use any text editor to edit Java and XML files then use command line tools to create, build and debug Android application

**About Native Code**

Libraries written in C and other languages can be compiled to ARM native code and installed, but the Native Development Kit is not yet officially supported by Google. Native classes can be called from Java code running under the Dalvik VM using the System. Load Library call, which is part of the standard Android Java classes.

**Creating an Android Project**

The ADT plug-in provides a New Project Wizard that you can use to quickly create a new Android project (or a project from existing code). To create a new project:

1. Select File > New > Project.
2. Select Android > Android Project, and click Next.
3. Select the contents for the project:
4. Enter a Project Name. This will be the name of the folder where your project is created.
5. Under Contents, select Create new project in workspace. Select your project workspace location.
6. Under Target, select an Android target to be used as the project's Build Target. The Build Target specifies which Android platform you'd like your application built against.
7. Unless you know that you'll be using new APIs introduced in the latest SDK, you should select a target with the lowest platform version possible, such as Android 1.1. Under Properties, fill in all necessary field.

**Enter an Application Name**: This is the human-readable title for your application—thename that will appear on the Android device.

1. Enter a Package name. This is the package namespace (following the same rules as for packages in the Java programming language) where all your source code will reside.
2. Select Create Activity (optional, of course, but common) and enter a name for your main Activity class.
3. Enter a Min SDK Version. This is an integer that indicates the minimum API Level required to properly run your application. Entering this here automatically sets the min SDK Version attribute in the <uses-sdk> of your Android Manifest file. If you're unsure of the appropriate API Level to use, copy the API Level listed for the Build Target you selected in the Target tab.
4. Click Finish.

Once you complete the New Project Wizard, ADT creates the following folders and files in your new project:

1. src/Includes your stub Activity Java file. All other Java files for your application go here.
2. <Android Version>/ (e.g., Android 1.1/) Includes the android.jar file that your application will build against. This is determined by the build target that you have chosen in the New Project Wizard.
3. gen/ This contains the Java files generated by ADT, such as your R.java file and interfaces created from AIDL files.
4. assets/ This is empty. You can use it to store raw asset files. See Resources and Assets.
5. res/ A folder for your application resources, such as drawable files, layout files, string values, etc. See Resources and Assets.
6. AndroidManifest.xml The Android Manifest for your project. See The AndroidManifest.xml File.
7. default. Properties This file contains project settings, such as the build target. This files is integral to the project, as such, it should be maintained in a Source Revision Control system. It should never be edited manually — to edit project properties, right-click the project folder and select "Properties".

**To create an AVD with the AVD Manager**

1. Select Window > Android SDK and AVD Manager, or click the Android SDK and AVD Manager icon (a black device) in the Eclipse toolbar.
2. In the Virtual Devices panel, you'll see a list of existing AVDs. Click New to create a new AVD.
3. Fill in the details for the AVD.
4. Give it a name, a platform target, an SD card image (optional), and a skin (HVGA is default).
5. Click Create AVD.

When you first run a project as an Android Application, ADT will automatically create a run configuration. The default run configuration will launch the default project Activity and use automatic target mode for device selection (with no preferred AVD).

**To Create or Modify a Launch Configuration**

Follow these steps as appropriate for your Eclipse version:

1. Open the run configuration manager.
2. In Eclipse 3.3 ,select Run > Open Run Dialog (or Open Debug Dialog)
3. In Eclipse 3.4 (Ganymede), select Run > Run Configurations (or Debug Configurations)
4. Expand the Android Application item and create a new configuration or open an existing one.

All the tips below follow from these two basic tenets:

Some would argue that much of the advice on this page amounts to "premature optimization." While it's true that micro-optimizations sometimes make it harder to develop efficient data structures and algorithms, on embedded devices like handsets you often simply have no choice. For instance, if you bring your assumptions about VM performance on desktop machines to Android, you're quite likely to write code that exhausts system memory. This will bring your application to a crawl — let alone what it will do to other programs running on the system!

That's why these guidelines are important. Android's success depends on the user experience that your applications provide, and that user experience depends in part on whether your code is responsive and snappy, or slow and aggravating. Since all our applications will run on the same devices, we're all in this together, in a way. Think of this document as like the rules of the road you had to learn when you got your driver's license: things run smoothly when everybody follows them, but when you don't, you get your car smashed up.

Before we get down to brass tacks, a brief observation: nearly all issues described below are valid whether or not the VM features a JIT compiler. If I have two methods that accomplish the same thing, and the interpreted execution of foo() is faster than bar(), then the compiled version of foo() will probably be as fast or faster than compiled bar(). It is unwise to rely on a compiler to "save" you and make your code fast enough.

**Avoid Creating Objects:**

Object creation is never free. A generational GC with per-thread allocation pools for temporary objects can make allocation cheaper, but allocating memory is always more expensive than not allocating memory.

If you allocate objects in a user interface loop, you will force a periodic garbage collection, creating little "hiccups" in the user experience.

Thus, you should avoid creating object instances you don't need to. Some examples of things that can help:

When extracting strings from a set of input data, try to return a substring of the original data, instead of creating a copy. You will create a new String object, but it will share the char[] with the data.

If you have a method returning a string, and you know that its result will always be appended to a String Buffer anyway, change your signature and implementation so that the function does then append directly, instead of creating a short-lived temporary object.  
A somewhat more radical idea is to slice up multidimensional arrays into parallel single one-dimension arrays:

An array of ints is a much better than an array of Integers, but this also generalizes to the fact that two parallel arrays of ints are also a **lot**more efficient than an array of (int,int) objects. The same goes for any combination of primitive types.

If you need to implement a container that stores tuples of (Foo,Bar) objects, try to remember that two parallel Foo[] and Bar[] arrays are generally much better than a single array of custom (Foo,Bar) objects. (The exception to this, of course, is when you're designing an API for other code to access; in those cases, it's usually better to trade correct API design for a small hit in speed. But in your own internal code, you should try and be as efficient as possible.) Generally speaking, avoid creatingshort-term temporary objects if you can. Fewer objects created mean less-frequentgarbage collection, which has a direct impact on user experience.

**CHAPTER-3**

**SYSTEM ANALYSIS**

Different software in the industry have different behaviours. That seeks out the necessity for the existing software to be functions in a particular operating system. This is analysed and made to work out the features and the software itself.

**3.1 EXISTING SYSTEM**

In the Existing System, there is everything that is needed to run an android application and is compatible with all the versions of the operating systems (above 2.3).

Since there are no special system calls to be made or system functions are to be revoked, there is no need to consider the issues of permissions by the Working Operating System.

**3.1.1 DRAWBACKS OF EXISTING SYSTEM**

i. The application is not compatible in every Android Devise.

ii. Requires a Internet source to update values.

To avoid these limitations and make the working more accurately this project helps.

**3.2 PROPOSED SYSTEM**

The aim of proposed system is to overcome the drawback in the existing system by going to develop this project which is compatible with almost every Android Device. And by improving the Android software we can also decrease the time that is excessively consumed in the existing systems.

**3.2.1 ADVANTAGES OF PROPOSED SYSTEM**

The system is very simple in design and to implement. The system requires very low system resources and the system will work in android phones/tablets. It has got following features.

1. Able to convert about 153 currencies from all around the world.
2. Gives up to date values by contacting the server. So there is negligible chance of causing or displaying any error.

**3.3 FEASIBILITY STUDY**

Preliminary investigation examines project feasibility the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All systems are feasible if they are given unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

* + Technical Feasibility
  + Operation Feasibility
  + Economical Feasibility

**3.3.1 TECHNICAL FEASIBILITY**

The technical issue usually raised during the feasibility stage of the investigation includes the following:

* Does the necessary technology exist to do what is suggested?
* Do the proposed equipments have the technical capacity to hold the data required to use the new system?
* Will the proposed system provide adequate response to inquiries, regardless of the number of users?
* Can the system be upgraded if developed?

**3.3.2 OPERATIONAL FEASIBILITY**

**User-friendly**

Customer will use the list of various currencies. Also the Customer wants the reports to view the various transactions based on the constraints. These modes are generated as user- friendly to the Client.

**Reliability**

The package will pick-up current currency values on line. Regarding the old values, User will enter them in to the system.

**Security**

The web server and database server should be protected from hacking, virus etc.

**Portability**

The application will be developed using standard open source software (Except Oracle) like Java, tomcat web server, Internet Explorer Browser etc. These software will work both on Windows and Linux o/s. Hence portability problems will not arise.

**Availability**

This software will be available always.

**Maintainability**

The system called the wheels uses the 1-tier architecture. It directly interacts with the user.

The front-end can be run on different system. Users access these forms by using different modes available

**3.3.3 ECONOMIC FEASIBILITY**

The computerized system takes care of the present existing system’s data flow and procedures completely and should generate all the reports of the manual system besides a host of other management reports.

It should be built as a web based application with separate web server and database server. This is required as the activities are spread throughout the organization customer wants a centralized database. Further some of the linked transactions take place in different locations. Open source software like TOMCAT, JAVA, MySQL and Linux is used to minimize the cost for the Customer.

**CHAPTER - 4**

**SOFTWARE REQUIREMENT SPECIFICATIONS**

Not all applications not on every operating system present out there. Every software has a tendency to work in only one particular operating system. The features of such operating system are mention in the following.

**4.1 FUNCTIONAL REQUIREMENTS**

**i. Dividing the Application among different Levels**

In this module, we need to set different levels of which every level has its own functionality. Preferences, update, timely update are the ones that are coded into the application.

**ii.** **Changing the Framework of the Application**

For some of the android based mobile phones, the framework done for a particular phone is not compatible. For that reason the framework of the Application for every android based mobile phone is necessary.

**4.2. NON FUNCTIONAL REQUIREMENTS**

The user interface is based on XML. The major non-functional requirements of the system are as follows.

**i. Usability**

The system is designed with completely automated process hence there is no or less user intervention.

**ii. Reliability**

The system is more reliable because of the qualities that are inherited from the chosen platform java .The code built by using java is more reliable.

**iii. Performance**

This system is developing in high level language and using the advanced front-end and back-end technologies it will give response to the end user on client system with in very less time.

**iv. Supportability**

The system is designed to be the cross platform supportable .The system us supported on a wide range of hardware and any software platform, which is having JVM .built into the system.

**v. Implementation**

The system is implemented in web environment and Windows 10 professional is used as the platform.

**4.3 PERFORMANCE REQUIREMENTS**

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in analysis of a system. Only when the requirements specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely with the users of the existing system to give the requirement specifications because they are people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. The requirement specification for any system can be broadly stated as given below. The system should be able to interface with the existing system.

•The system should be accurate

•The system should be better than the existing system

The existing system is completely dependent on the user to perform all the duties.

**4.4 SOFTWARE REQUIREMENTS**

|  |  |  |
| --- | --- | --- |
| **4.4.1 For Development** | |  |
| Operating System : | | Windows XP/2003/7/8/8.1/10 |
| Coding Language : | | Java 1.6/1.7/1.8 |
| Technology Used : | | Android 2.3 and above |
| IDE | : | Android Studio 1.0/1.4 |
| Plug-ins | : | SDK android plugin |
| Tools used | : | Android SDK 1.2, Google API |
| **4.4.1 For Deployment** | |  |
| Android OS | : | Android 2.3 Gingerbread and above |

**4.5 HARDWARE REQUIREMENTS**

|  |  |  |
| --- | --- | --- |
| **4.5.1 For Development** | |  |
| System | : | Pentium 2.4 GHz |
| Hard Disk | : | 40GB |
| RAM | : | 1 GB and above. |
| **4.4.1 For Deployment** | |  |
| Android Mobile with: | |  |
| CPU | : | 1 GHz and above |
| RAM | : | 512 MB and above |
| Memory | : | 2 MB [.apk file size] |

**Android Studio with SDK Manager**

The Software Development Kit (SDK) plug-in for Android Studio adds powerful extensions to the Android Studio integrated development environment. It allows you to create and debug Android applications easier and faster. If you use the Studio, the SDK plug-in gives you an incredible boost in developing Android applications:

It gives you access to other Android development tools from inside the Studio IDE. For example, SDK lets you access the many capabilities of the DDMS tool: take screenshots, manage port-forwarding, set breakpoints, and view thread and process information directly from Eclipse.It provides a New Project Wizard, which helps you quickly create and set up all of the basic files you'll need for a new Android application. It automates and simplifies the process of building your Android application.

The Android SDK includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator (based on QEMU), documentation, sample code, and tutorials. Currently supported development platforms includex86-architecture computers running Linux (any modern desktop Linux distribution), Mac OS X 10.4.8 or later, Windows XP or Vista. The officially supported integrated development environment (IDE) is Eclipse (3.2 or later) using the Android Development Tools (ADT).

**CHAPTER - 5**

**SYSTEM DESIGN**

A system is configured in such a way that it has the ability to perform functions and operations effectively and with advantages. For the configuration, every entity in the development cycle has to make use of some pre-existing diagrams and methods.

**5.1 MODULE DESCRIPTION**

Nowadays people have lots of works to manage thus are in a state of thinking all the time due to which they tend to forget a lot of things of which phone is a major one. At times people forget it at some random corner of the office or home and in such cases there are chances that the phone is left in silent profile which makes it difficult for the user to find it.

So to solve the above mentioned issue this app will include 2 major modules:

**5.2 DATA FLOW DIAGRAMS**

A data-flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).

On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process. The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done further explosion is necessary and an adequate amount of detail is described for analyst to understand the process. A

DFD is also known as a “bubble chart” has the purpose of clarifying system requirements

and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

**5.3 UML DIAGRAMS**

**Unified Modelling Language**:

The Unified Modelling Language allows the software engineer to express an analysis model using the modelling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

**i.User Model View**

This view represents the system from the user’s perspective. The analysis representation describes a usage scenario from the end-users perspective.

**ii.Structural model view**

In this model the data and functionality are arrived from inside the system. This model view models the static structures.

**iii.Behavioural Model View**

It represents the dynamic of behavioural as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

**iv.Implementation Model View**

In this the structural and behavioural as parts of the system are represented as they are to be built.

**v.Environmental Model View**

In this the structural and behavioural aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

* UML Analysis modelling, this focuses on the user model and structural model views of system.
* UML design modelling, which focuses on the behavioural modelling, Implementation modelling and environmental model views.

Use case Diagrams represent the functionality of the system from a user’s point of view.

Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behaviour of the system from external

Point of view. Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer etc., or another system like central database.

**Components and Types**

Design of a system consists of classes, interfaces and collaboration. UML provides class diagram, object diagram to support this. Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support implementation perspective. Process defines the flow of the system. UML includes the following diagrams and the details are described:

* Class diagram
* Use case diagram
* Sequence diagram
* Collaboration diagram
* Activity diagram
* State chart diagram
* Deployment diagram
* Component diagram

**5.3.1 Use Case Diagram**

In software and system’s engineering, a **use case** is a list of steps, typically defining Interactions between a role and a system, to achieve a goal. The actor can be a human, an external system, or time.

In systems engineering use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. As an important requirement technique, use cases have been widely used in modern software engineering over the last two decades. Use case driven development is a key characteristic of process models and frameworks like Unified Process (UP), Rational Unified Process (RUP), Oracle Unified Method (OUM), etc. With its iterative and evolutionary nature, use case is also a good fit for agile development.

**Contents**

* Actors
* Use cases
* Their Relationships.

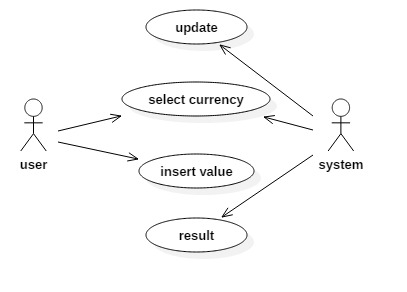
The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

**Purpose**

The purpose of use case diagram is to capture the dynamic aspect of the system. But this description is too generic to describe the purpose. Because other four diagrams (activity, sequence, collaboration, state chart) is also having the same purpose. So we will look in to some specific purpose to distinguish it from other four diagrams. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analysed to gather its functionality use cases are prepared and actors are identified.

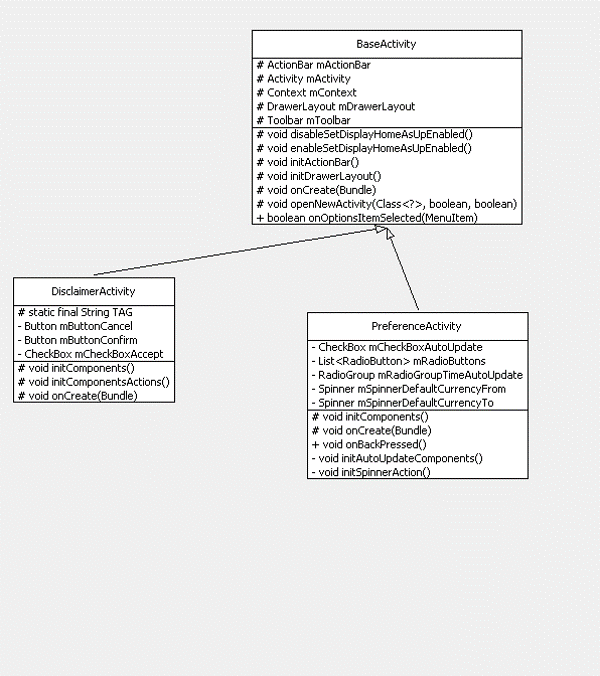
So in brief, the purpose of use case diagrams is as follows:

* Used to gather requirements of a system.
* Used to get an outside view of a system.
* Identify external and internal factors influencing the system.
* Show the interacting among the requirements are actors.



**Fig.5.1 Use Case Diagram**

**5.3.2 Class Diagram**



**Fig.5.2 Class Diagram**

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when de signing a system, conceptual, specification, and implementation.

**Purpose**

The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used at the time of construction. The UML diagrams like activity diagram, sequence diagram can only give the sequence diagram can only give the sequence flow of the application but class diagram is a bit different. So it is the mostpopular UML diagram in the coder community. So the purpose of the class diagram can be summarized as:

* Analysis and design of the static view of an application.
* Describe responsibilities of a system.
* Base for component and deployment diagrams.
* Forward and reverse engineering.

**Components**

* Classes
* Interfaces
* Collaborations
* Dependency, Generalization and Association Relationships.

**5.3.3 Sequence Diagram**

**Purpose**

1. To capture dynamic behavior of the system.
2. To describe the message flow in the system.
3. To describe structural organization of the objects.
4. To describe interaction among objects.

**Contents of Sequence Diagram**

* Objects
* Flow of control
* Messages
* Life line

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

UML sequence diagrams are used to represent the flow of messages, events and actions between the objects or components of a system. Time is represented in the vertical direction showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram.

Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario. UML sequence diagrams are useful design tools because they provide a dynamic view of the system behaviour which can be difficult to extract from static diagrams or specifications.

**i.Actor**

Represents an external person or entity that interacts with the system.

**ii.Object**

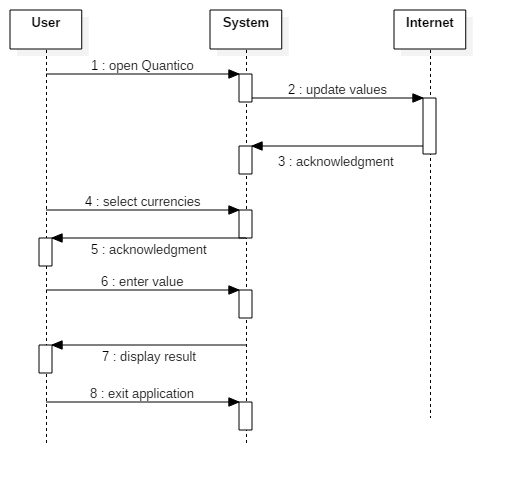
Represents an object in the system or one of its components.

**iii.Separator**

Represents an interface or boundary between subsystems, components or units (e.g., air interface, Internet, network).

**iv.Group**

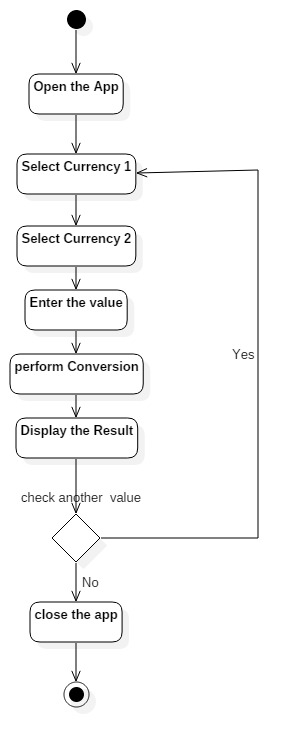
Groups related header elements into subsystems or components.



**Fig.5.3 Sequence Diagram**

**5.3.4 Activity Diagrams**

Activity diagrams describe the workflow behaviour of a system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel.



**Fig.5.4 Activity Diagram**

**Contents**

* Initial/Final state
* Activity
* Fork and Join
* Branch
* Swim lanes

**Fork**

A fork represents the splitting of the single flow of control in to two or more concurrent flow of control. A fork may be one incoming transition and two or more outgoing transitions, each of which represents an independent flow of control. Below fork the activities with each of these path continues in parallel.

**Join**

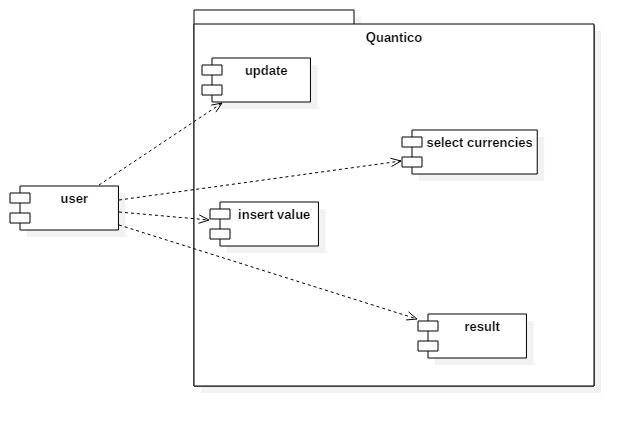
A join represents the synchronization of two or more concurrent flow of control. A join may have two or more incoming transition and one outgoing transition. Above the join the activities associated with each of these paths continues in parallel.

**Branching**

A branch specifies the alternate paths takes based on some Boolean expression. Branch is represented by diamond. Branch may have one incoming transition and two or more outgoing one on each outgoing transition, you place a Boolean expression shouldn’t overlap but they should cover all possibility.

**5.3.5 Component Diagram**

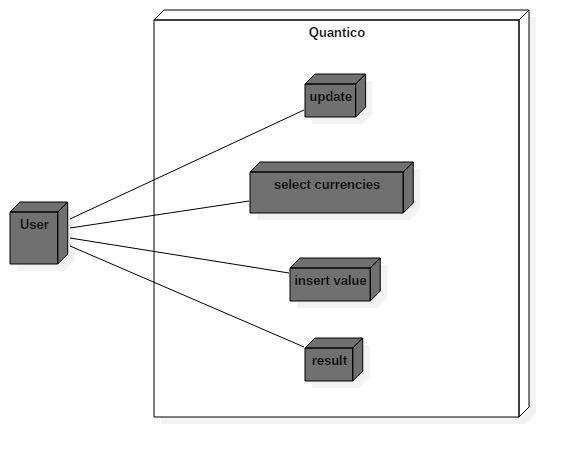
Component diagrams show the software components of a system and how they are related to each other. These relationships are called dependencies. Component diagrams can also show the interfaces used by the components to communicate to each other. The component diagram contains components and dependencies. Components represent the physical packaging of a module of code. The dependencies between the components show how changes made to one component may affect the other components in the system. Dependencies in a component diagram are represented by a dashed line between two or more components.



**Fig.5.5 Component Diagram**

**5.3.6 Deployment Diagram**

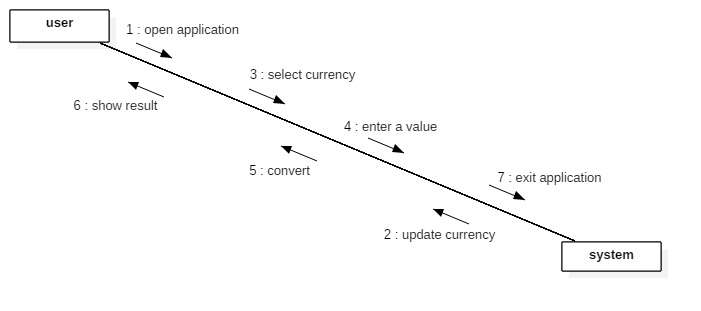
Deployment diagrams show the physical relationship between hardware and software in a system. The deployment diagram contains nodes and connections. The combined deployment and component diagram below gives a high level physical description of the completed system. A node usually represents a piece of hardware in the system. A connection depicts the communication path used by the hardware to communicate and usually indicates a method.



**Fig.5.6 Deployement Diagram**

**5.3.7 Collaboration Diagram**

Like sequence diagrams, collaborationdiagrams are also interactiondiagrams. Collaborationdiagrams convey the same information as sequence diagrams, but focus on object roles instead of the times that messages are sent. In a sequence diagram, object roles are the vertices and messages are the connecting links

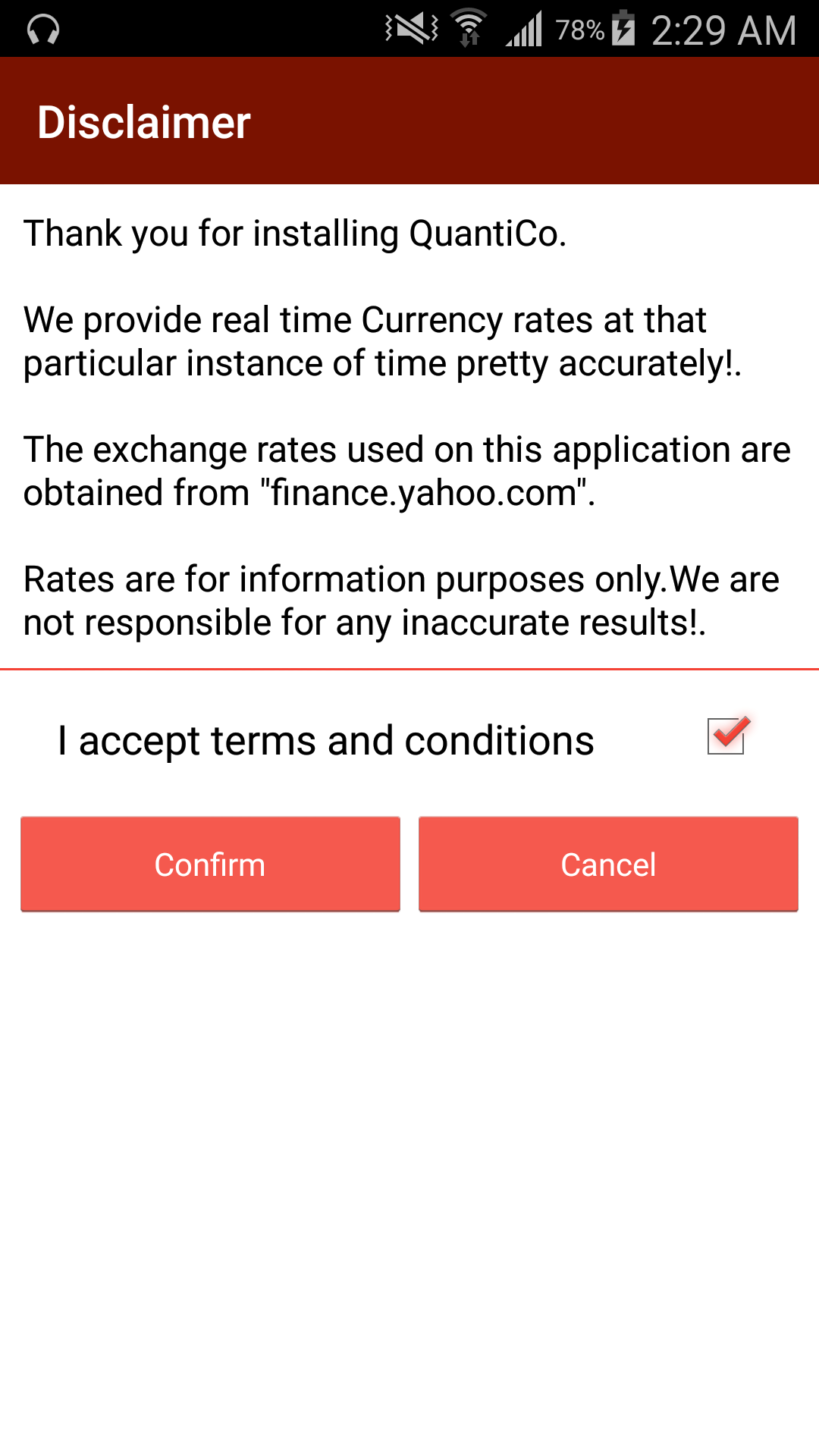


**Fig.5.7 Collaboration Diagram**

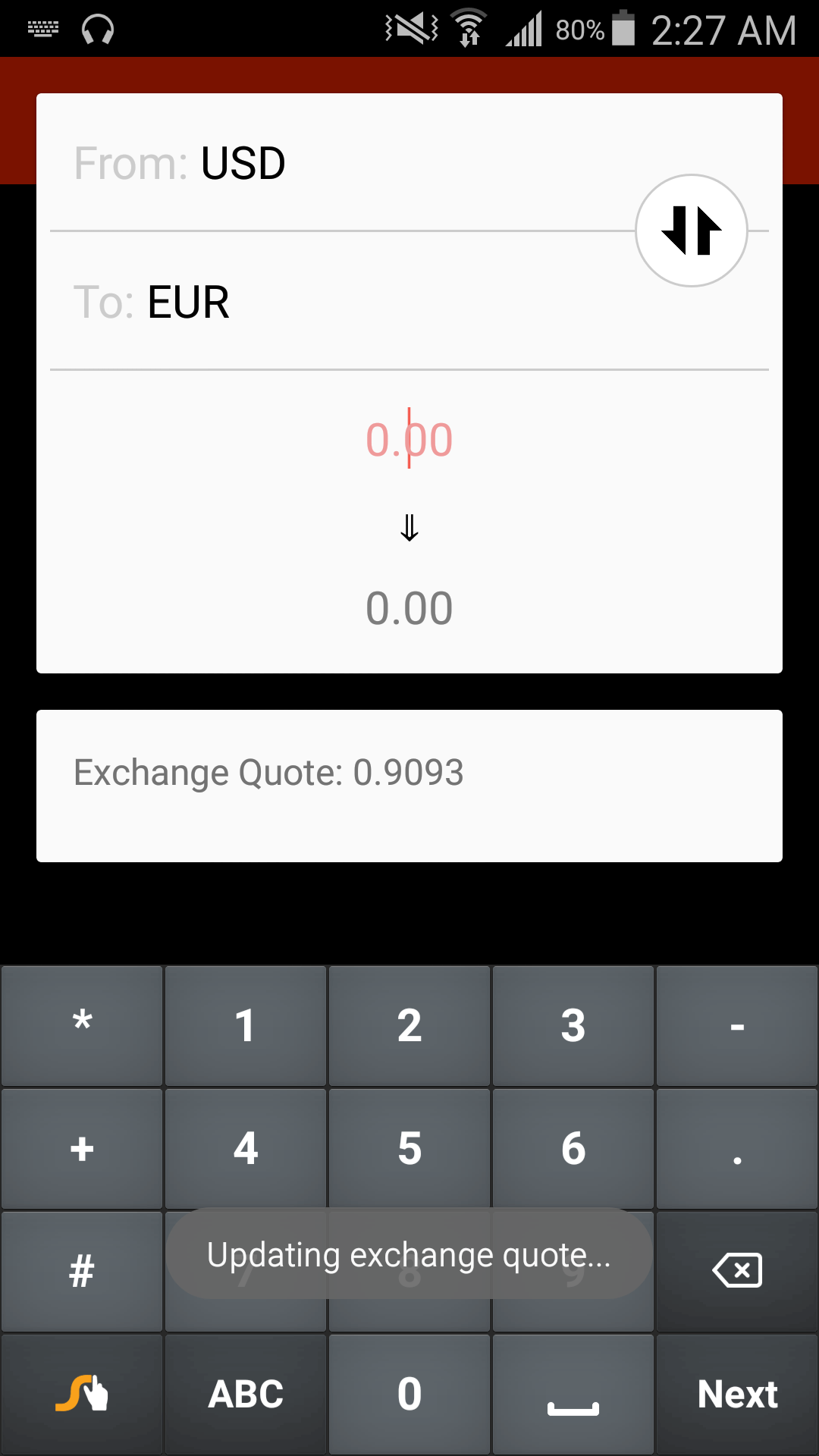
**CHAPTER-6**

**IMPLEMENTATION**

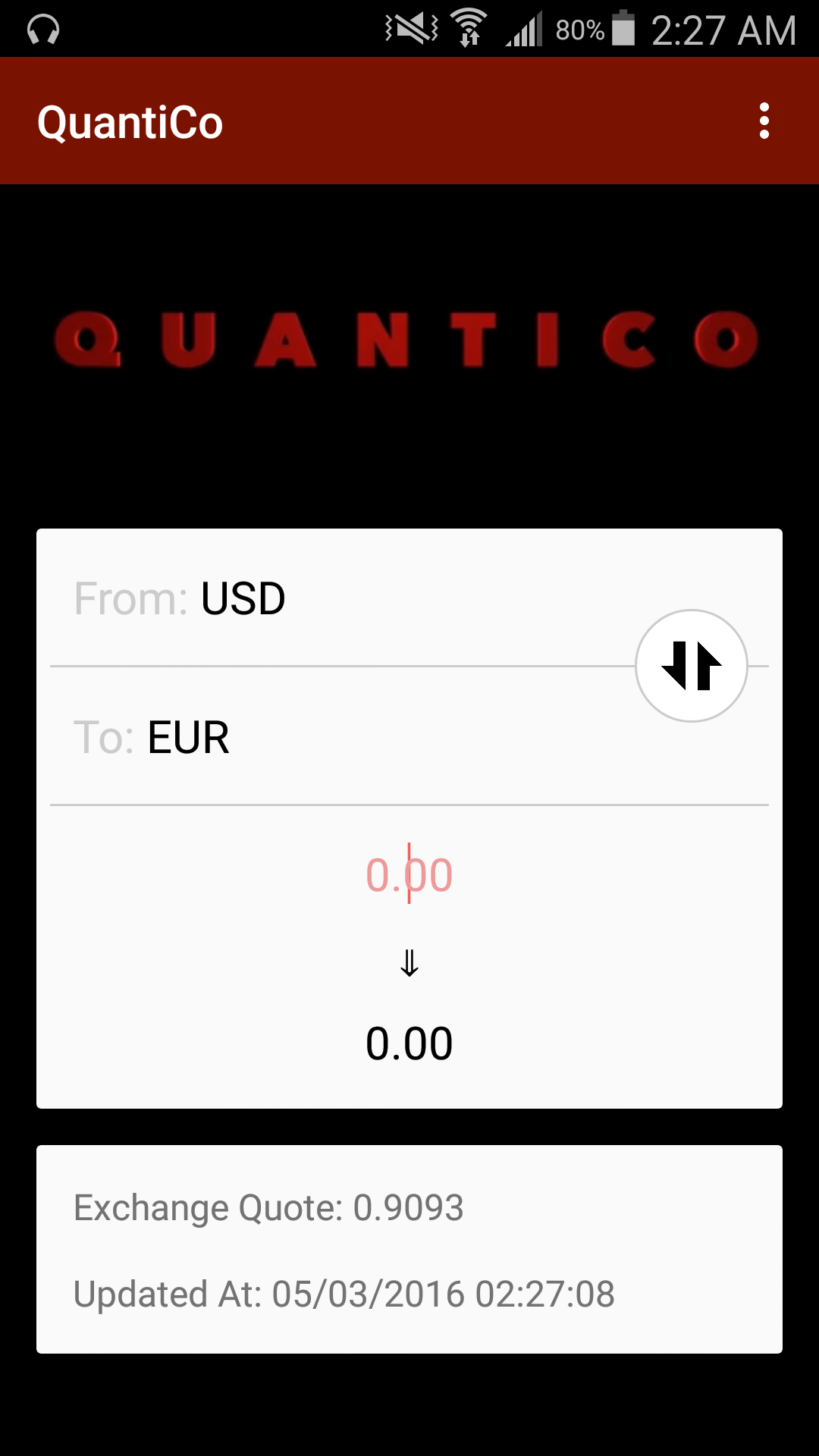
**6.1 SCREENSHOTS:**



**Fig.6.1 Disclaimer Screenshot**



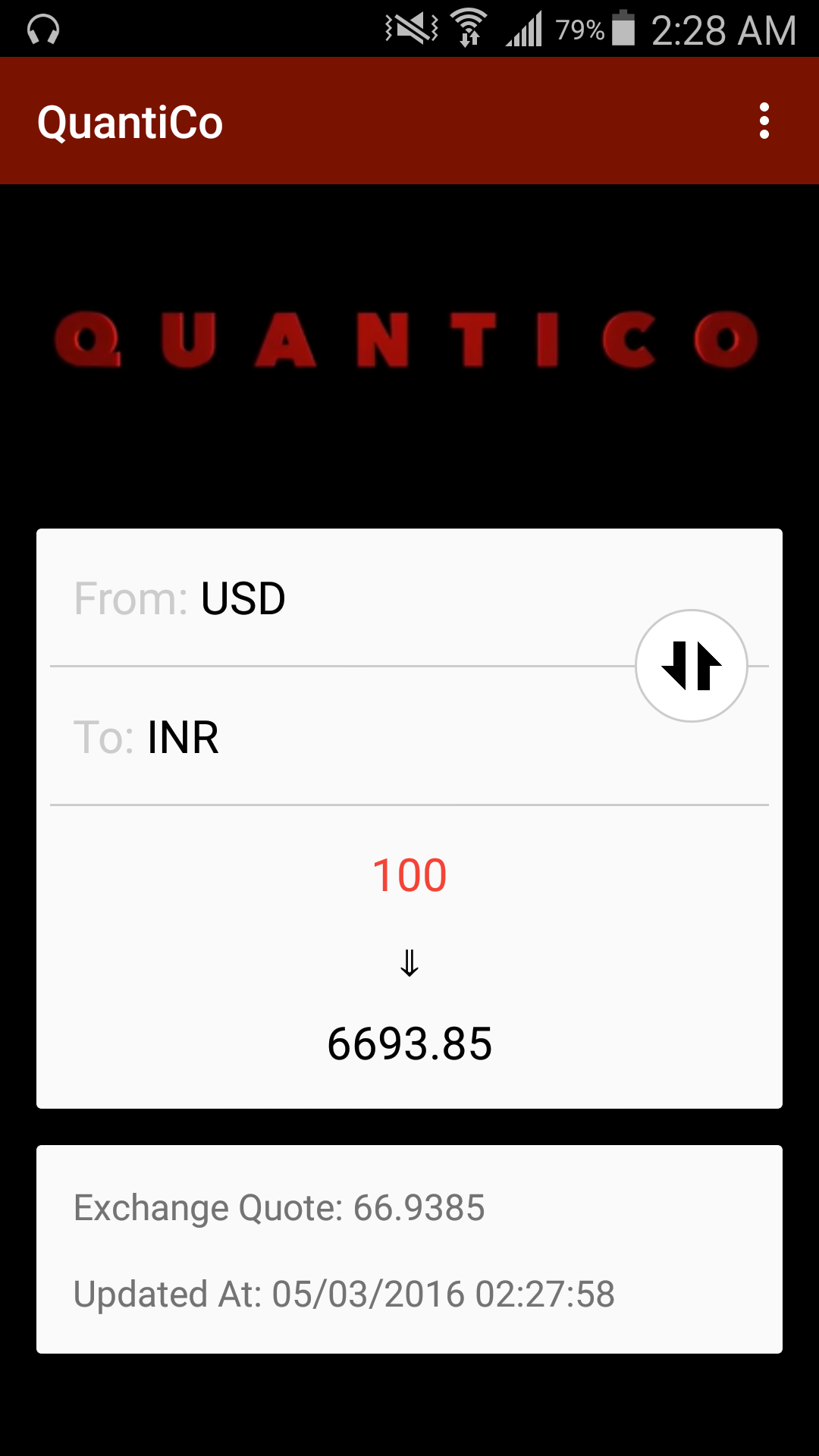
**Fig.6.2 updating Screenshot**



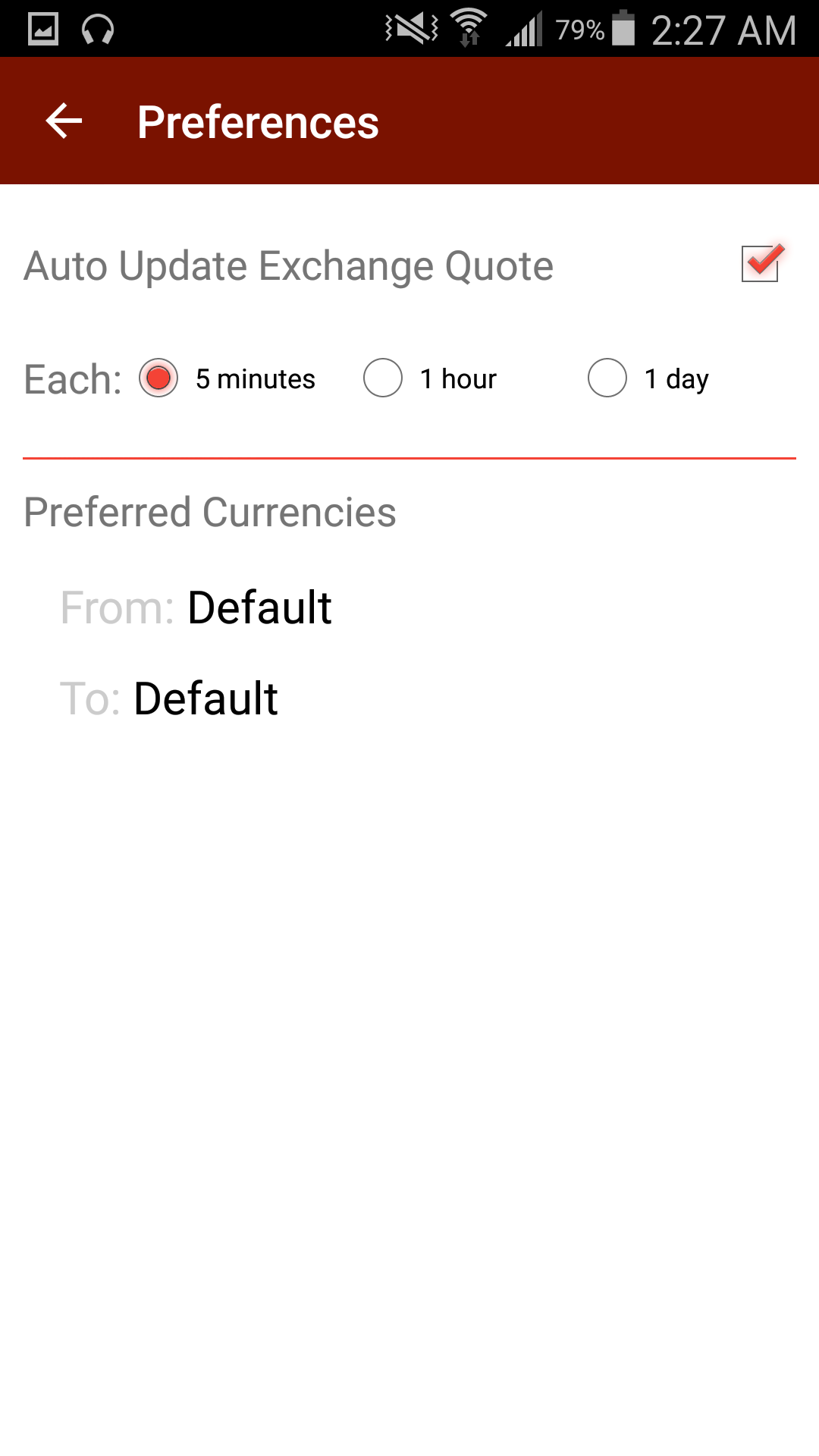
**Fig.6.3 Home Screenshot**



**Fig.6.4 Drop Down Currencies Screenshot**



**Fig.6.5 Converted Screenshot**

****

**Fig.6.6 Preference Screenshot**

**CHAPTER-7**

**SYSTEM DESIGN**

**7.1 INTRODUCTION TO TESTING**

**Software testing** is an investigation conducted to provide stakeholders with informationabout the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects)

It involves the execution of a software component or system to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

1. meets the requirements that guided its design and development,
2. responds correctly to all kinds of inputs,
3. performs its functions within an acceptable time,
4. is sufficiently usable,
5. can be installed and run in its intended environments, and
6. Achieves the general result its stakeholder’s desire. As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects).
   1. **TEST CASES**

The development of software systems involves of a series of production activities where opportunities for injection of human fallibilities are enormous. Errors may begin to occur at the very inception of the process where the objectives may be erroneously or imperfectly specified, as well as in later design and development stages.

**7.2.1 Testing types**

The following are the types of testing

* Unit testing
* Integration testing
* Validation testing
* System testing

**Unit Testing:**

Unit testing focuses verification effort on the smallest unit of software design (ie.) the module. Unit testing exercise specific paths in a module’s control structure to ensure complete coverage and maximum error detection .This test focus on each module individually ensure that it functions properly as a unit. Hence, the name is unit testing.

**Integration Testing:**

Integration testing address the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of High-order tests are conducted. The main objectives in this testing process is to take unit tested modules and build a program structure that has been dictated by design.

The following are the types of Integration Testing

**Top-Downs Integration:** This method is an incremental approach to theconstruction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first of breadth-first manner.

**Bottom-Up integration:** This method begins the construction and testing with themodules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is elimination. The bottom-up integration strategy may be implemented with the following steps

* The low-level modules are combined into clusters that perform a Specific software sub-function.
* A drive (i.e.,), the control program for testing is written to co-ordinate test Case input and output.
* The cluster is tested.
* Drivers are removed and clusters are combined moving upward in the Program structure.

**Validation Testing:**

At the end of the Integration Testing, software is completely assembled as a package, interfacing errors have been uncovered and correction testing begins.

**Validation Test Criteria**

Software testing and validation is achieved through serried of black box tests that demonstrate conformity with the requirements. A test plan outlines the classes of tests to be conducted and a test procedure defines specific test cases that will be used to demonstrate conformity with requirements. Both, the plan and the procedure are designed to ensure that all functional requirements are achieved, documentation is correct and other requirements are met.

**System Testing:**

System testing is series of different tests whose primary purpose is to fully exercise the computer based system. Although each test has a different purpose, all the work should verify that all system element have been properly integrated and perform allocated functions. FWC has gone through all these tests, and it is bug free.

**CHAPTER-8**

**CONCLUSION**

This application known as Quantico is the web for android platform. This information and education based on the application in which the user can manipulate the retrieved data is effective and user-friendly at the same time. This application is been for android mobile using android.

This application contains the details such as 'from' currency, 'to' currency and the resultant values at real time.

A currency converter stores the most recent market valuations of the world's currencies, which allows individuals to compare the value of one currency against those of others in the database.

The values of the different currencies are determined based on the supply or demand of dealing prices.

User to system communication is very friendly which means any person can use it with ease. This would be an effective application for the conversion of currencies.

It is an application that is designed to look in android operating system. The user interface deals with a set of instructions from the user. This is done the given value into the value of the currency of user's choice.

Time and resources are also effectively used and that makes this application very reliable and productive.

**REFERENCES**

The below given textbooks and links are referred for the development of this project:

[1] Object oriented analysis and design by Grady Booch.

[2] Herbert Scheldt 2008,”Java Complete Reference”, Tata McGraw-Hill, 7th Edition.

1. Android Programming Big Nerd Ranch Guide.pdf
2. Grady Booch, James Rambaugh.1998, “Unified Modelling Language User Guide”.
3. http://developer.android.com/guide/index.html
4. Software Engineering A Practitioner‘s Approach-Seventh Edition by Roger S. Pressman
5. Programming Android by G. Blake Meike
6. Applying UML AND Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development by Craig Larman
7. Amazing Android Apps for Dummies by Daniel A. Begun