# BUDDY LOCATOR

**APPLICATION DEVELOPMENT – I**

**SOFTWARE REQUIREMENT SPECIFICATION**

*Submitted in partial fulfilment for the award of the degree*

*of*

**Master of Technology (M.Tech)**

***in***

**Information Technology**

*by*

**KARANDEEP SINGH SALUJA**

**(REG NO 14MIN2658)**

*Under the guidance of*

**Prof. P.S RAMESH**

**PROFESSOR**

**VIT University**


**School of Information Technology and Engineering**

**Mar, 2017**

# Contents



# Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
|---|---|---|---|
| 0.1 | Karandeep Singh | Initial version for Buddy Locator | 18/03/2017 |

# 1 Introduction

The project which we are going to discuss through this document is the "Buddy Locator" – an Android application intended to be used by anyone to locate their friends and family in real-time using GPS.

In this section, we will see a brief introduction on the product scope, mainly on the purpose of the document and the readers.

## 1.1 Purpose

This SRS describes the software functional and non-functional requirements for release 1.0 of the Buddy Locator. This document is intended to be used by the members of the project team that will implement and verify the correct functioning of the system. Unless otherwise noted, all requirements specified here is high priority and committed for release 1.0.

## 1.2 Product Scope

The scope of the Buddy Locator will be to provide the functionality of finding your friends and family in real-time using GPS. The goal and objective of the project is to provide the users with easy access for locating their buddy whenever and wherever they want.

Benefits of the system would be for both friends and family.

Friends can get location of their buddies and easily track each other. They do not need to guide them to come where they are already present. It is also helpful in case of family to keep track of their family members and keep them safe and tracking.

It is also helpful just in case anyone of the friends or family get into some danger. Using this app, we can easily trace them and locate.

Even while travelling during nights, in case anyone gets into trouble they can use SOS functionality to send emergency notification to the defined emergency contact numbers mentioned in the app.

## 1.3 Definitions, Acronyms and Abbreviations

BL – Buddy Locator

SOS – Emergency Mode

## 1.4 References and Acknowledgments

- IEEE SRS format
- Project Specification Requirement ( provided by IBM )
- http://www.bradapp.com/docs/sdd.html
- https://web.cs.dal.ca/~arc/teaching/CS3130/Templates/Design%20Templates/Software%20Design%20Specification.doc

## 1.5 Overview

The SRS will include two sections,

namely:

- **Overall Description:** This section will describe major components of the system, interconnections, and external interfaces.
    - **Specific Requirements:** This section will describe the functions of actors, their roles in the system and the constraints faced by the system.

# 2 General Description

## 2.1 Product Perspective

Buddy Locator will be to provide the functionality of finding your friends and family in real-time using GPS. The goal and objective of the project is to provide the users with easy access for locating their buddy whenever and wherever they want.

Benefits of the system would be for both friends and family.
Friends can get location of their buddies and easily track each other. They do not need to guide them to come where they are already present. It is also helpful in case of family to keep track of their family members and keep them safe and tracking.

It is also helpful just in case anyone of the friends or family get into some danger. Using this app, we can easily trace them and locate.

Even while travelling during nights, in case anyone gets into trouble they can use SOS functionality to send emergency notification to the defined emergency contact numbers mentioned in the app.

## 2.2 Product Functionality

The major and common functionality of the product will be as below:

**Account creation:**
Any Applicant can create account by providing mobile number for registration.

**Authentication**:
A user will get OTP for the first time verification after which he can set the password for the same.

**Authorization**:

Each Application will access the app with their mobile number and password. They will send request to their contacts for access to their location. Only if they approve or accept the request only then they will be able to see the location of each other.

## 2.3 User Characteristics

Buddy Locator should support the following users:

- Applicant

**Applicant:**

The User should be able to do the following operations:

- Sign in to the App
- Sign up to the App if logging for the first time.
- Select the contact from his phone and ask for location access
- Approve the request of any friend or family for location access
- Access SOS mode for sending emergency notification when in problems.
- Log out from the App

## 2.4 General Constraint

**Client Requirement:**

- The application is expected to work in a minimum of Android Version 4.0 (Icecream Sandwich).
- Hardware Limitations: The minimum hardware requirement for the phone is 1 GB RAM and 150 MB ROM minimum and a GPS enabled smartphone.
- Accessibility: Initially, the software should be available as an android application for a small set of users to test.
- Others: The application should be built using Java, and it should, initially, be accessible through Android Studio and later published on a server.

**Connectivity:**

- The client (smartphone) should always be connected with Data (Internet)
- Client should always have GPS turned-on.

## 2.5 Assumptions and Dependencies

**Assumptions:**

- User Interface: The type of client interface (front-end) to be supported is – GUI based
- The scope of the application is limited to acceptance and approval of sharing request.
- It will use google maps service to locate the friends and family.
- GPS is always in turned-on mode.

**Dependencies:**

- The operation of the Buddy Locator depends totally on Internet availability, GPS service being on and acceptance of location sharing.

**Expectations:**

- The application should be designed in a social and parenting domain.
- The application should provide real-time locations given the GPS is turned on.
- The application should also update the last available location just in case the GPS is turned off.
- The app should support multitasking like seeing multiple friend's location.
- Compilation and Build should be done using Android Studio.
- Source-code and all documents must be maintained (checked-in) in configuration management system (subversion).
- Wipro's coding standards (for Java) should be followed.
- Deliverables should include use-case diagrams, design document, compiled and tested source code, test-plans, test-cases documents, test-results and release note.

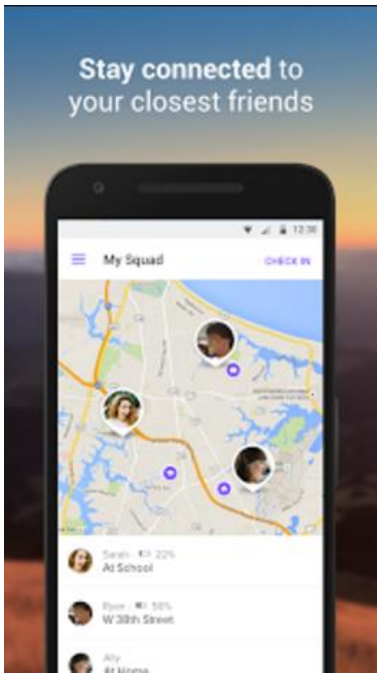
# 3 Specific Requirements

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

The Applicant will be able to sign in to the app. There will be different activity for Sign In and Sign Up.

The Buddy Locator will look somewhat like below.



It will locate the friends or family in the map like below:

In case of Panic, SOS button can be pressed as below:


It will send us a notification saying your particular friend or family needs help.

### 3.1.2 Hardware Interfaces

The basic hardware component that the system requires would be an android smartphone with minimum of version 4.1 (Icecream Sandwich).

The physical and logical characteristics of the machine includes size, RAM, ROM, Internet Service and GPS. All these hardware components in the smartphone make its functioning fast and effective.

The system will be developed on Android Studio using Java technology.

Apart from the hardware component above, no other hardware interfaces are identified in the system.

### 3.1.3 Software Interfaces

**Software Interface**

- Front End Client: Android Studio
- Middleware : Java
- Database : SQLite
- Backend: Java

## Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

## Project Structure

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under Gradle Scripts and each app module contains the following folders:

- manifests: Contains the AndroidManifest.xml file.
- java: Contains the Java source code files, including JUnit test code.
- res: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select Project from the Project dropdown (in figure 1, it's showing as Android).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the Problems view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

## Activities

Activities are one of the fundamental building blocks of apps on the Android platform. They serve as the entry point for a user's interaction with an app, and are also central to how a user navigates within an app (as with the Back button) or between apps (as with the Recents button).

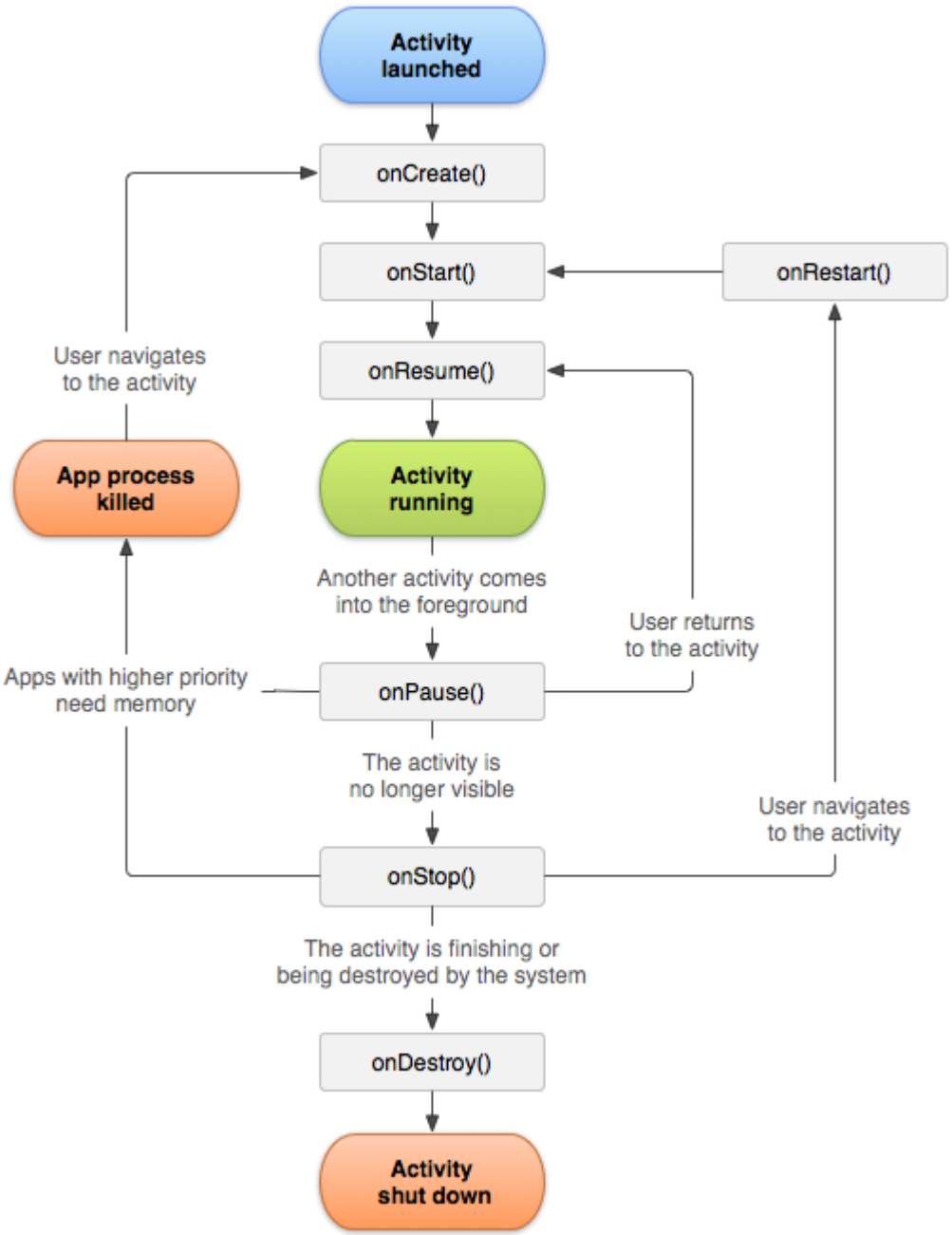
Skillfully managing activities allows you to ensure that, for example:

- Orientation changes take place smoothly without disrupting the user experience.
- User data is not lost during activity transitions.
- The system kills processes when it's appropriate to do so.

## Life Cycle of Activity

To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: onCreate(), onStart(),onResume(), onPause(), onStop(), and onDestroy(). The system invokes each of these callbacks as an activity enters a new state.

**Figure 1.** A simplified illustration of the activity lifecycle.

- 


As the user begins to leave the activity, the system calls methods to dismantle the activity. In some cases, this dismantlement is only partial; the activity still resides in memory (such as when the user switches to another app), and can still come back to the foreground. If the user returns to that activity, the activity resumes from where the user left off. The system's likelihood of killing a given process—along with the activities in it—depends on the state of

the activity at the time. Activity state and ejection from memory provides more information on the relationship between state and vulnerability to ejection.

Depending on the complexity of your activity, you probably don't need to implement all the lifecycle methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

The next section of this document provides detail on the callbacks that you use to handle transitions between states

## Architecture of Android

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.

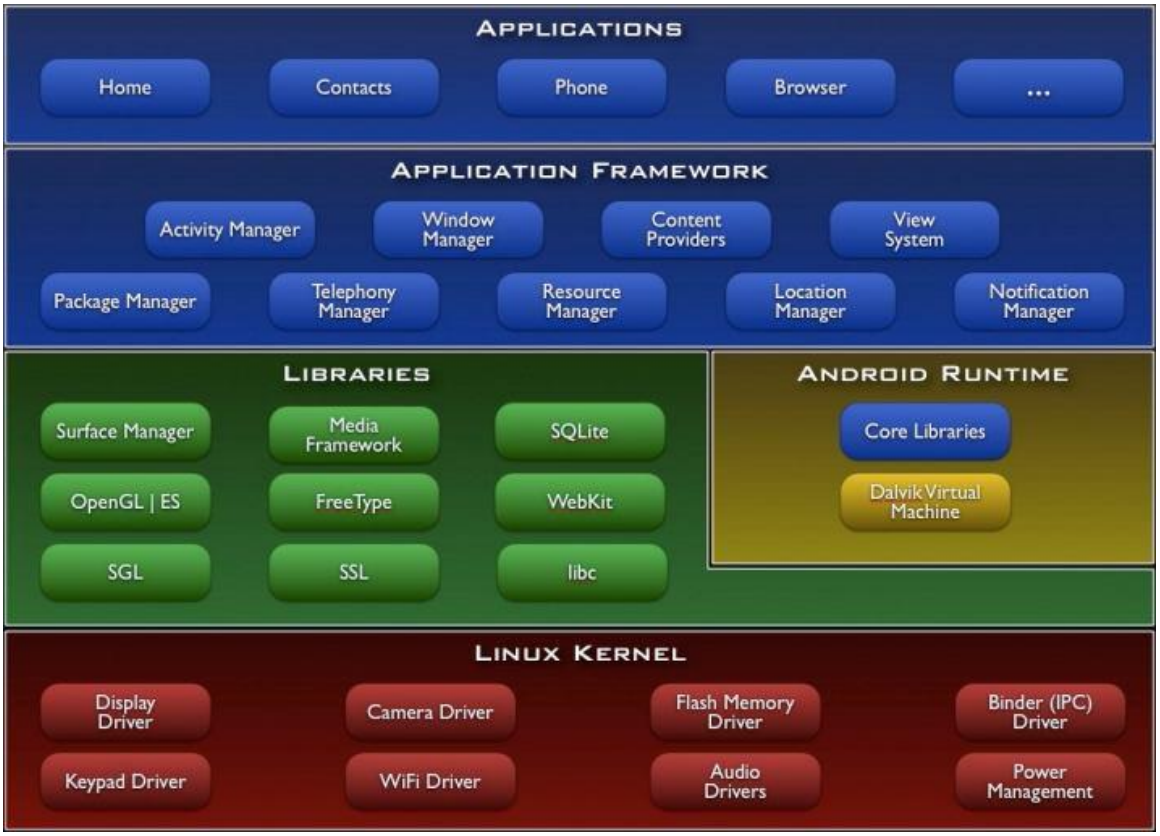



**Fig 4.2.2: Android Architecture**

## Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

## Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

## Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −

android.app − Provides access to the application model and is the cornerstone of all Android applications.

android.content − Facilitates content access, publishing and messaging between applications and application components.

android.database − Used to access data published by content providers and includes SQLite database management classes.

android.opengl − A Java interface to the OpenGL ES 3D graphics rendering API.

android.os − Provides applications with access to standard operating system services including messages, system services and inter-process communication.

android.text − Used to render and manipulate text on a device display.

android.view − The fundamental building blocks of application user interfaces.

android.widget − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.

android.webkit − A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

### 3.1.4 Communications Interfaces

- The Buddy Locator shall send a message notification to the user to verify the authenticity of the user itself.
- The Buddy Locator shall send a message notification to the user to accept the location request, if any of it is pending.
- The Buddy Locator shall send a message notification to the user confirming if the request is accepted by the friend or family.

As part of communication interfaces, there are several websites providing APIs for OTP verification like 2factor etc.

For OTP verification, Use of those APIs can be done.

## 3.2 Functional Requirements

The users of the Buddy Locator are:

Applicant: Can sign up and use it with others friends who are registered with the app for location exchanges.

Below are the requirements specific to the types of users:

**Applicant:**

| Requirement ID | Description |
|---|---|
| US-001 | The applicant should be able to send request for location sharing to friends and family |

| | |
|---|---|
| US-002 | The applicant should be able to view the location of his or her friends or family, given they have accepted to share their locations. |
| US-003 | The applicant should be able to enter his / her details to maintain his profile |
| US-004 | The applicant should be able to send SOS notification to his or her emergency contacts. |
| US-005 | The applicant should be able to see the last available location just in case the GPS is turned off. |

**Authentication & Authorisation:**

| **Requirement ID** | **Description** |
|---|---|
| AA-001 | All users logging into the system should be authenticated using mobile number and OTP for first time authentication after which they can set a password for their login. |

**Exit Handlers:**

| **Requirement ID** | **Description** |
|---|---|
| EX-001 | The application should handle logout just in case the user wants to log out from the app completely. |

## 3.3 Use Cases

### Overview

To model a system the most important aspect is to capture the dynamic behaviour. To clarify a bit in details, dynamic behaviour means the behaviour of the system when it is running /operating.

So only static behaviour is not sufficient to model a system rather dynamic behaviour is more important than static behaviour. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So use case diagrams are consists of actors, use cases and 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.

So to model the entire system numbers of use case diagrams are used.

### Purpose

The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because other four diagrams (activity, sequence, collaboration and State chart) are also having the same purpose. So we will look into some specific purpose which will distinguish it from other four diagrams.

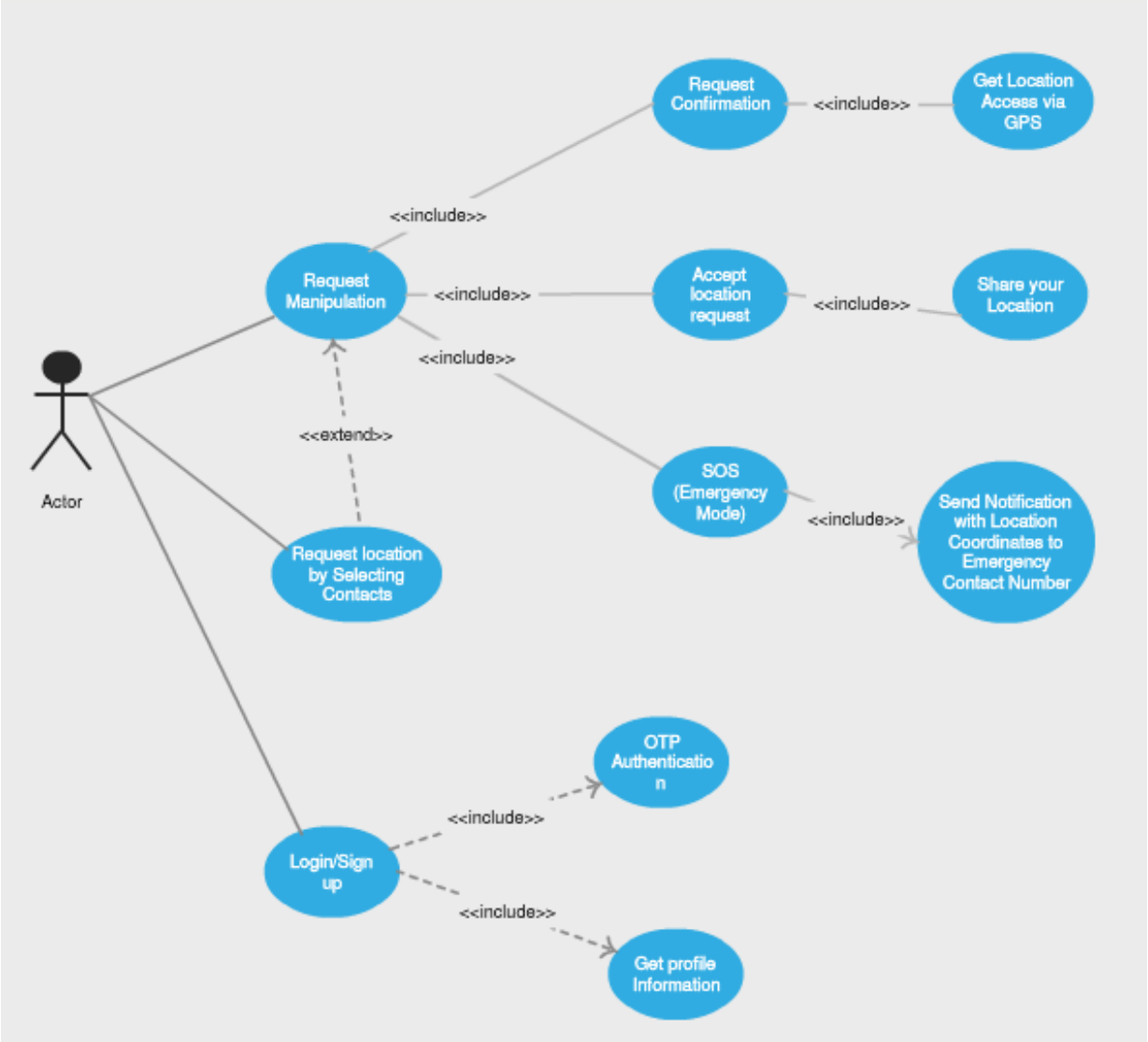
Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modelled to present the outside view.

So in brief, the purposes of use case diagrams can be 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.

Applicant use case view:



## 3.4 Classes/Objects

### Classes

The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages. The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a *structural diagram*.

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 flow of the application but class diagram is a bit different. So it is the most popular UML diagram in the coder community.

Class diagrams are the most popular UML diagrams used for construction of software applications. So it is very important to learn the drawing procedure of class diagram.

### Classes

Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

Object diagrams are used to render a set of objects and their relationships as an instance.

The purpose of a diagram should be understood clearly to implement it practically. The purposes of object diagrams are similar to class diagrams.

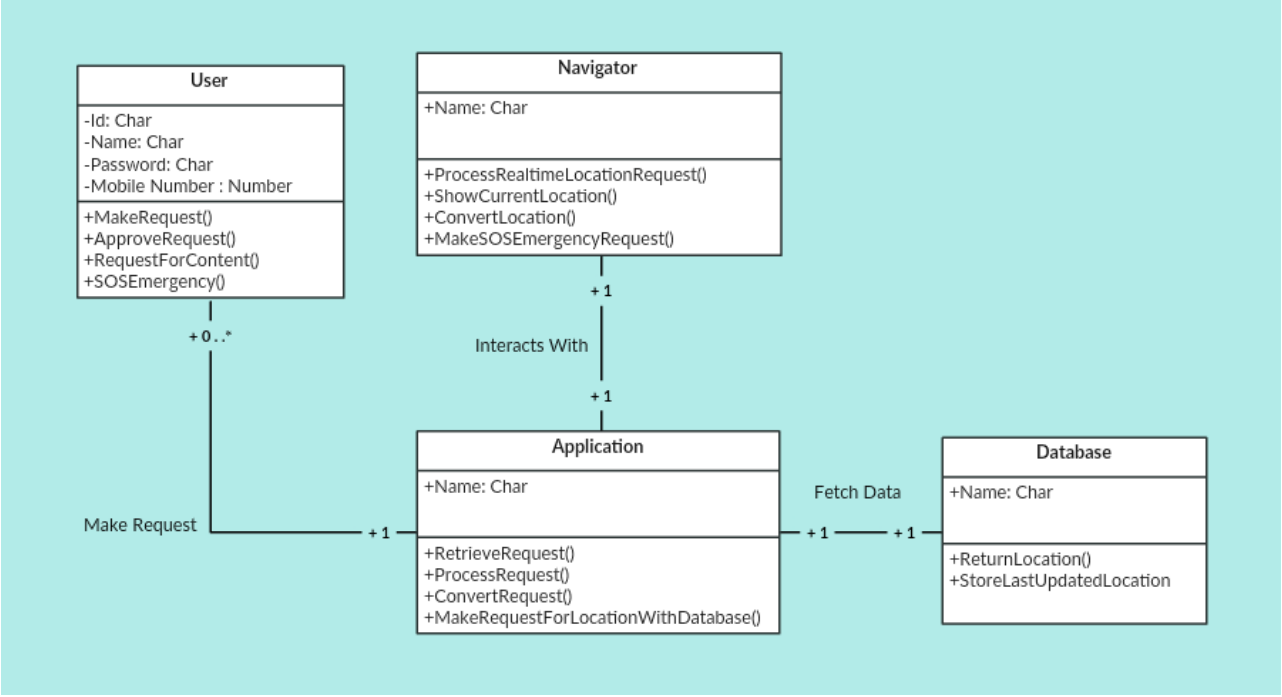
The difference is that a class diagram represents an abstract model consisting of classes and their relationships. But an object diagram represents an instance at a particular moment which is concrete in nature.

It means the object diagram is more close to the actual system behaviour. The purpose is to capture the static view of a system at a particular moment.

So the purpose of the object diagram can be summarized as:

- Forward and reverse engineering.
- Object relationships of a system
- Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective

**Overall flow:**



## 3.5 Non-Functional Requirements

### 3.5.1 Performance Requirements

Performance Requirements include:

- Activity should be loaded to the user within 10 seconds to avoid frustration to the user.
- Image display should be accurate and appropriate to be displayed.
- Any transaction / navigation should not take more than 15 seconds.
- Syncing with google maps should happen within 15 seconds.
- Any downtime / server issues / network issues should be promptly informed to the user.
- The application should be intuitive to use and easy-to-maintain.
- The application should be able to handle simultaneous requests from multiple clients.
- The system shall accommodate 400 users with estimated average session duration of 8 minutes.
- All activity generated by the system shall be fully loadable in no more than 10 seconds over a 40KBps modem connection.
- The system shall display confirmation messages to users within 4 seconds after the user submits information to the system.

### 3.5.2 Reliability

- The software system could provide automatically generated backup (on emulates storage) containing all the stored information at the time the backup is taken. The system shall allow authorized users to restore the data from an existing backup.

### 3.5.3 Availability:

The Buddy Locater shall be available to users on the App Store with full guide.

### 3.5.4 Safety and Security Requirements

Safety and security requirements include:

- Upon installation, the application will ask for runtime permissions.
- The user has to allow the app with all the runtime permissions in order for it to work properly
- After giving the permissions, User must sign in or sign up to the application, in order to access it.
- Confidential information such as user's password, his/her account information should be handled securely to avoid integrity issues.
- The APIs to be used securely for OTP authentication and verification of the user.

### 3.5.5 Maintainability

The application will be easy to maintain and be fully functional even when other application upgrades occur. The system will perform with normally under any conditional state.

### 3.5.6 Portability

Application will be available on internet, anyone can access that in the Android Smartphone with minimum of Version 4.1(Icecream Sandwich). Customers just need to have internet access.

## 3.6 Inverse Requirements

Inverse requirements describe the constraints on allowable behaviour. In many cases, it is easier to state that certain behaviour must never occur than to state requirements guaranteeing acceptable be-heavier in all circumstances. Software safety and security requirements are frequently stated in manner.

## 3.7 Design Constraint

Specific accounting standards, ISO policies for the centre can constrain the software product. Extensive database capabilities are also limited.

## 3.8 Logical Database Requirements

This section explains the data storage requirements of the Buddy Locator and ***indicative*** table (database) structure. The following sections explain few of the tables required for the application and the other tables will have to be designed accordingly.

### *Table: User Information*

The user specific details such as name, phone number, authentication and authorization / privileges should be kept in one or more tables, as necessary and applicable.

Table 1: User Details

| **Field Name** | **Data type** | **Description** |
|---|---|---|
| *PhoneNo* | VARCHAR(10) | Phone No of the user. It will also act as a user id |
| *First Name* | VARCHAR(15) | First Name of the user |
| *Last Name* | VARCHAR(15) | Last Name of the user |
| *EmailID* | VARCHAR [50] | Email ID of user for communication |
| *Password* | VARCHAR(15) | Password of user |

### *Table: Authentication Information*

The table contains Authentication information for Administrator.

Table 2: LoginCredentials

| **Field Name** | **Data type** | **Description** |
|---|---|---|
| PhoneNo | VARCHAR(10) | User Identification |
| Password | VARCHAR(9) | Password |

***Table: Transaction Information***

The table contains Transaction Information.

Table 3: LastLocationUpdate

| **Field Name** | **Data type** | **Description** |
|---|---|---|
| UpdateNo | VARCHAR(10) | Update number of location |
| LastUpdated | Date | Date of last updated location |
| Location | VARCHAR | Location of the user |

## 3.9 Other Requirements

Database: All the data will be stored in a relational database.

Dynamic Requirements specify constraints on the execution characteristics of the system. They typically include response time and throughout of the system. Since these factors are not applicable to the proposed software, it will suffice if the response tine is high and the transactions are carried out precisely and quickly.

# 4 Analysis Model

## 4.1 Sequence Diagrams

A **Sequence diagram** is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. 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**.

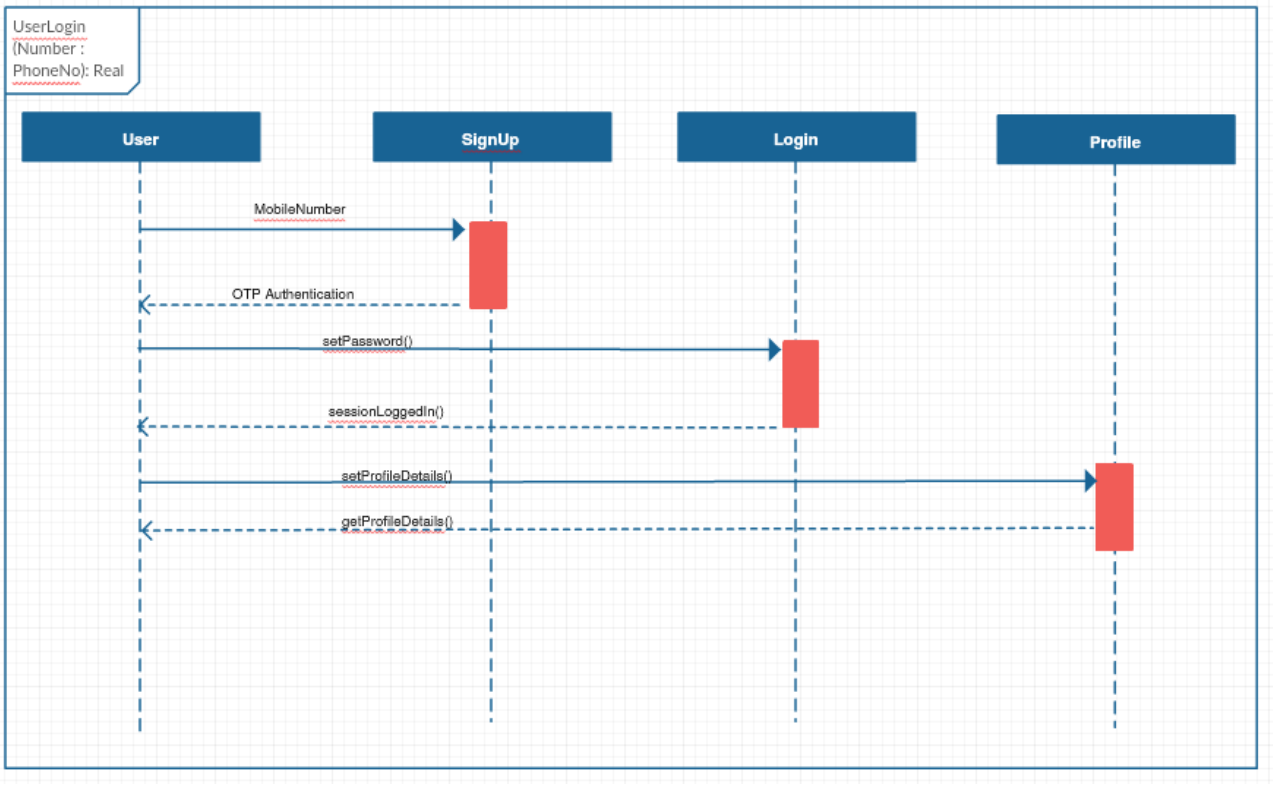
A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

If the lifeline is that of an object, it demonstrates a role. Leaving the instance name blank can represent anonymous and unnamed instances. Messages written with horizontal arrows with the message name written above them, display interaction. Solid arrow heads represent synchronous calls, open arrow heads represent asynchronous messages, and dashed lines represent reply messages. If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response. Asynchronous calls are present in multithreaded applications and in message-oriented middleware. Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (Execution Specifications in UML).

Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing.

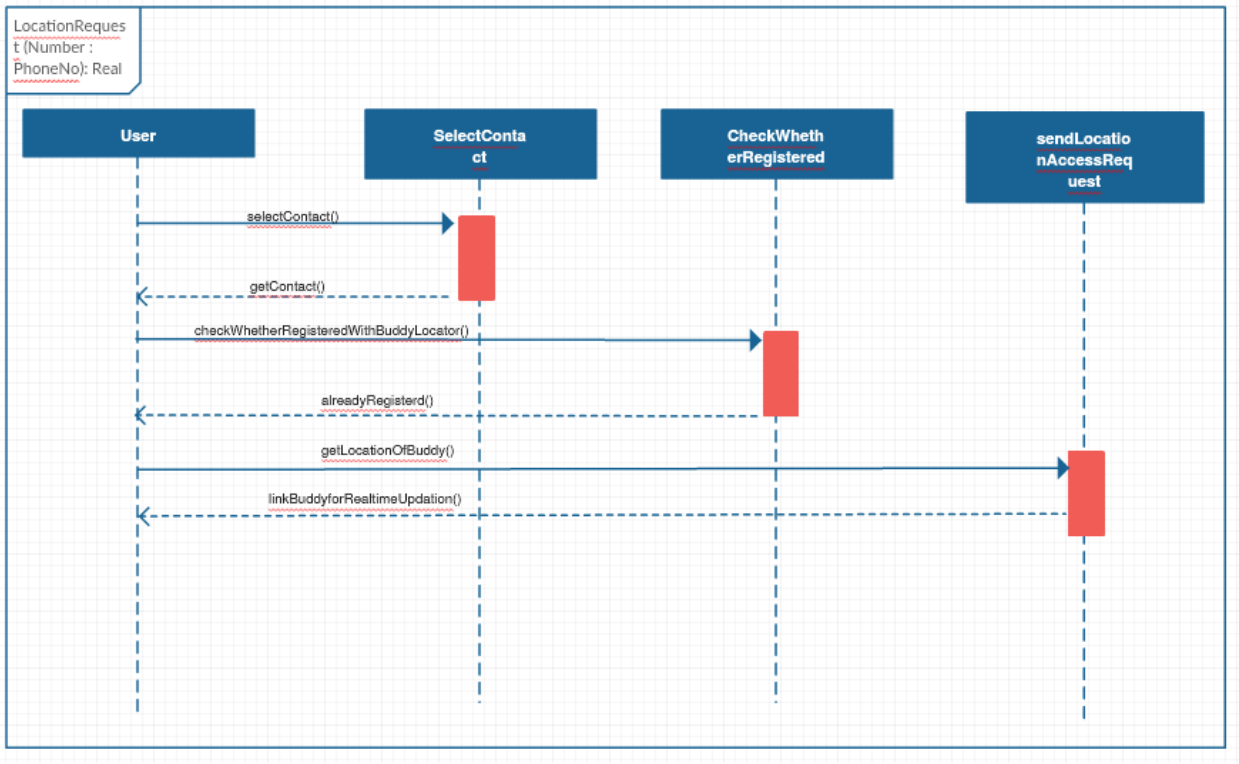
When an object is destroyed (removed from memory), an X is drawn on top of the lifeline, and the dashed line ceases to be drawn below it (this is not the case in the first example though). It should be the result of a message, either from the object itself, or another.

## User Login:



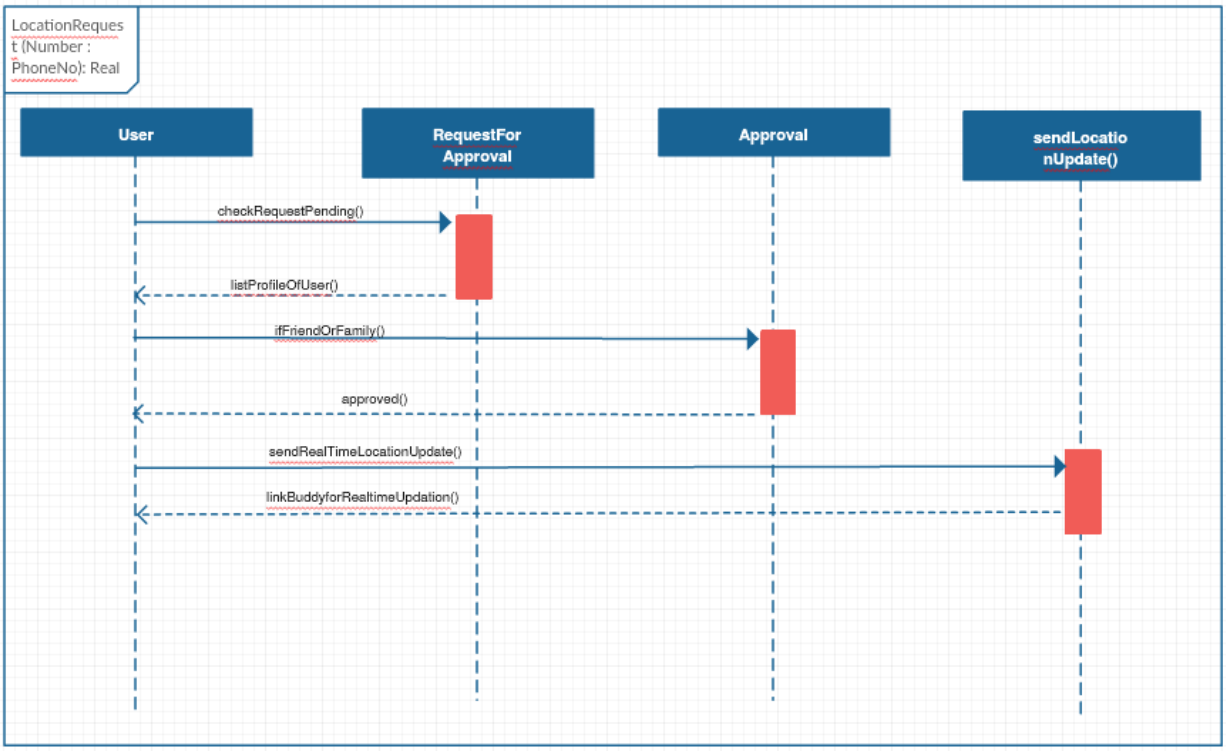

This sequence diagram depicts the steps involved in the registeration and logging of a user in the application.

## Location Request:



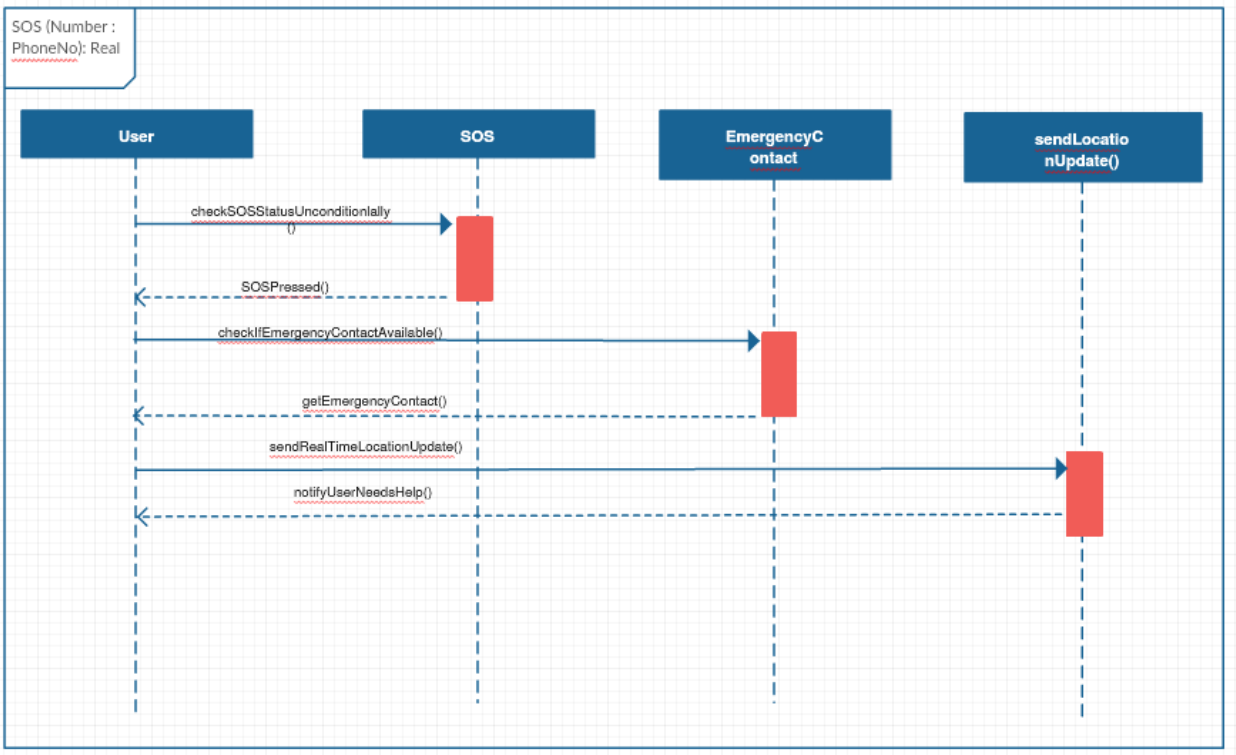

This sequence diagram depicts the sending of location access request to any of your contacts and once accepted, link the friend or family as a buddy.

## Location Access Approval



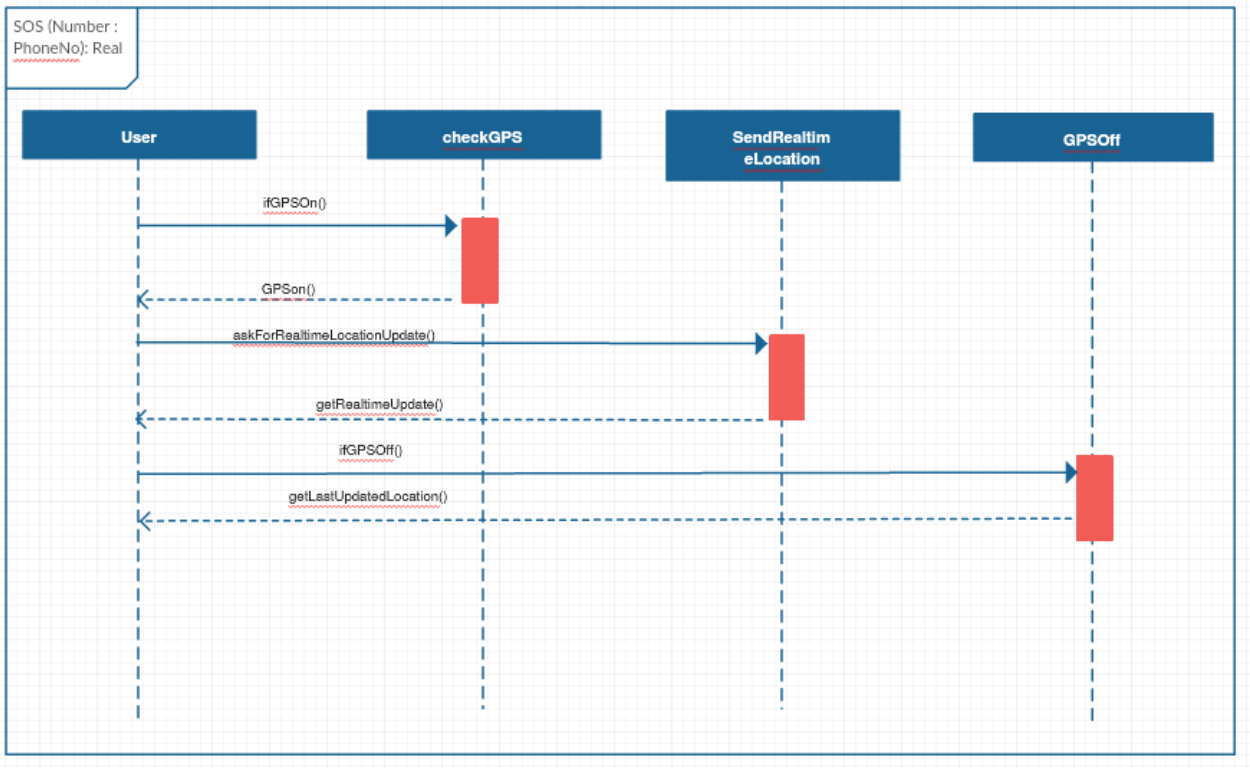

This sequence diagram depicts the approval of location access request from any of our contacts and once accepted, link the friend or family as a buddy.

## SOS:




This sequence diagram depicts the working of SOS. It keeps on checking for SOS. Once SOS event occurs, it checks for emergency contact number saved, if any. It takes all the emergency contacts(Max 3) and sends the location update notification to them with emergency notification.

## Last Location Update




This sequence diagram depicts if the GPS is turned off. When the GPS is turned off, it checks for the last location update and send those coordinates to the user asking for its buddy details.

## 4.2 State-Transition Diagrams (STD)

State transition diagrams have been used right from the beginning in object-oriented modelling. The basic idea is to define a machine that has a number of states (hence the term finite state machine). The machine receives events from the outside world, and each event can cause the machine to transition from one state to another. For an example, take a look at figure 1. Here the machine is a bottle in a bottling plant. It begins in the empty state. In that state it can receive squirt events. If the squirt event causes the bottle to become full, then it transitions to the full state, otherwise it stays in the empty state (indicated by the transition back to its own state). When in the full state the cap event will cause it to transition to the sealed state. The diagram indicates that a full bottle does not receive squirt events, and that an empty bottle does not receive cap events. Thus you can get a good sense of what events should occur, and what effect they can have on the object.

## How to create State Transition Diagram

Statechart diagram is used to describe the states of different objects in its life cycle. So the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

Statechart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

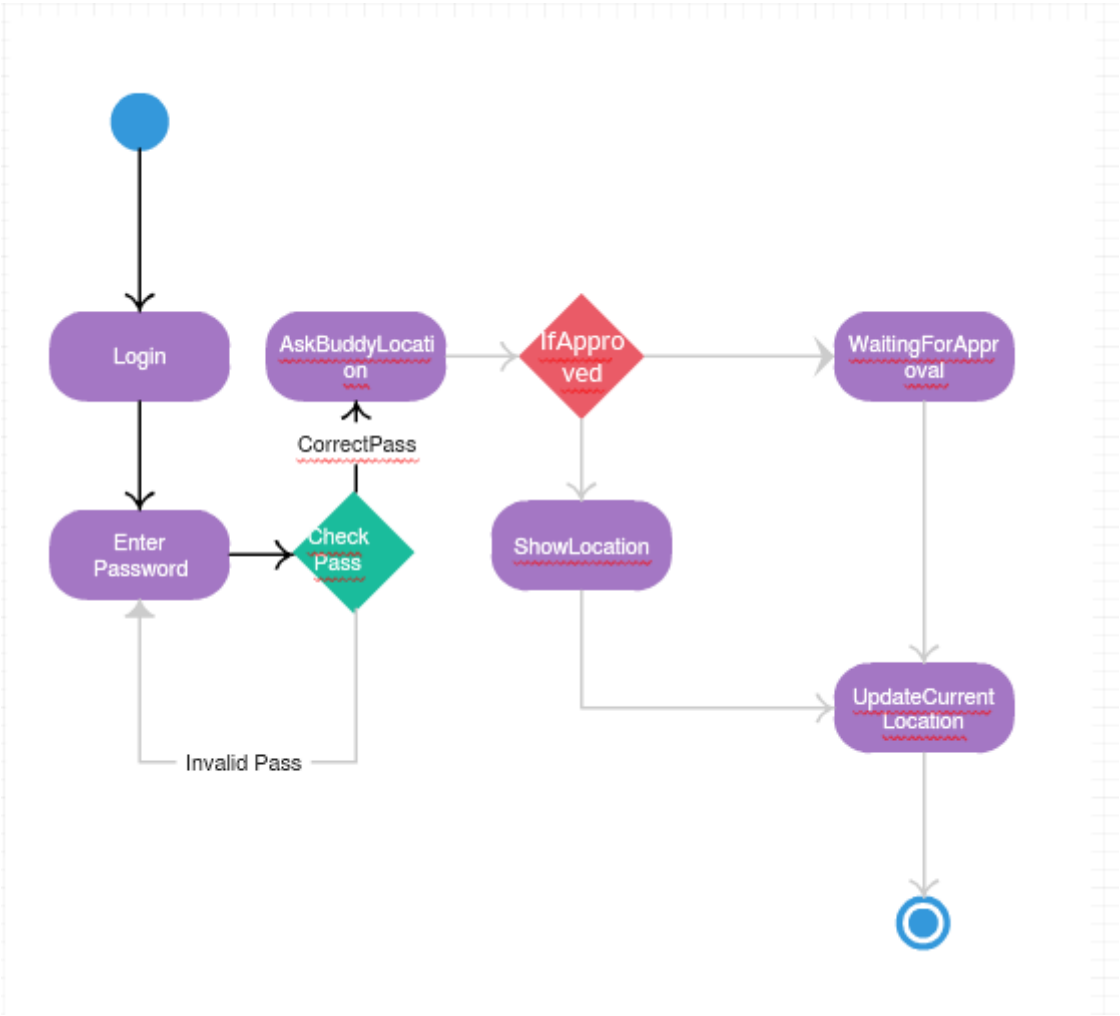
Before drawing a Statechart diagram we must have clarified the following points:

- Identify important objects to be analyzed.
- Identify the states.
- Identify the events.

The following is an example of a Statechart diagram where the state of Order object is analyzed.

The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for state changes of order object.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exists also. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete it is considered as the complete transaction



## 4.3 Data Flow Diagrams (DFD)

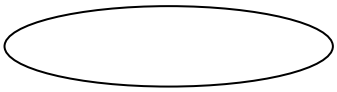
A dataflow diagram modules a system by using external entities from which data flow to a process which transforms the data and creates output data flow which go other processes or external entities or data source. Store data may also be used as an input. The main merit of DFD is that it can provide an overview of what data a system would process, use and where the result flow.

DFDs are structure in such a way that starting from a diagram which gives a broad overview at glance, they can be expended to hierarchy of diagrams giving more and more details.

**Symbols used in DFDs**

There are 4 symbols used in Data Flow Diagrams

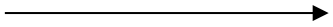
- A circle is used to depict process



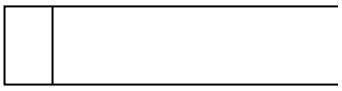
- A rectangle represent the external entities.

- Data flow is shown by arrows

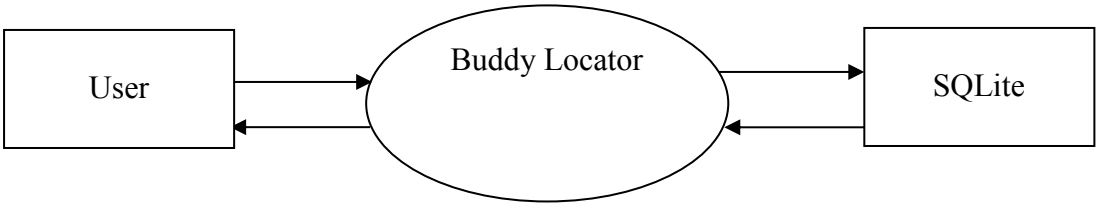
s



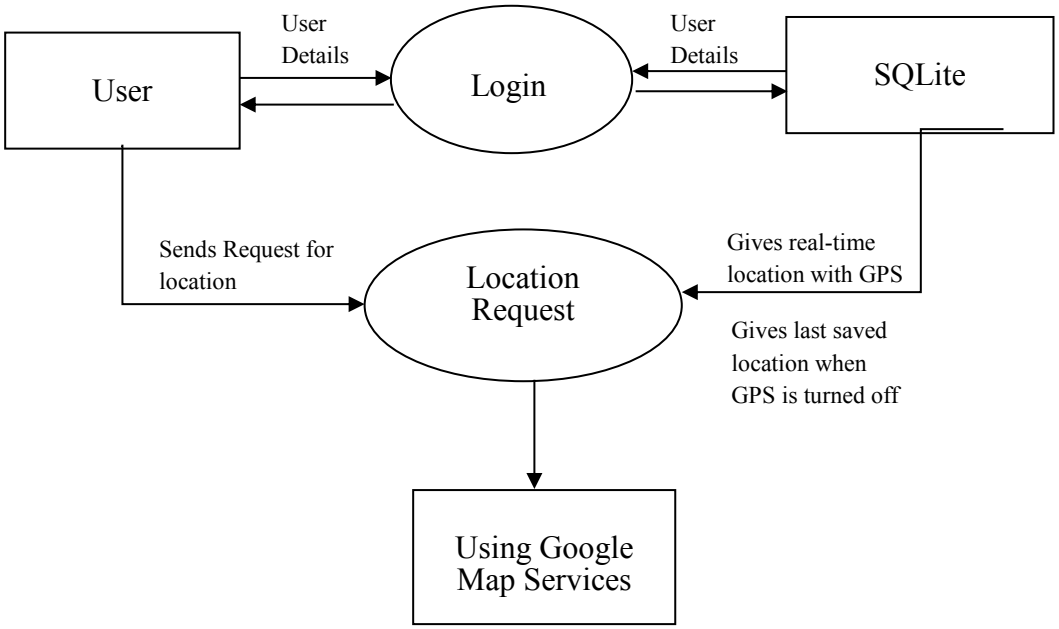
- The inventory storage is shown by double line as shown



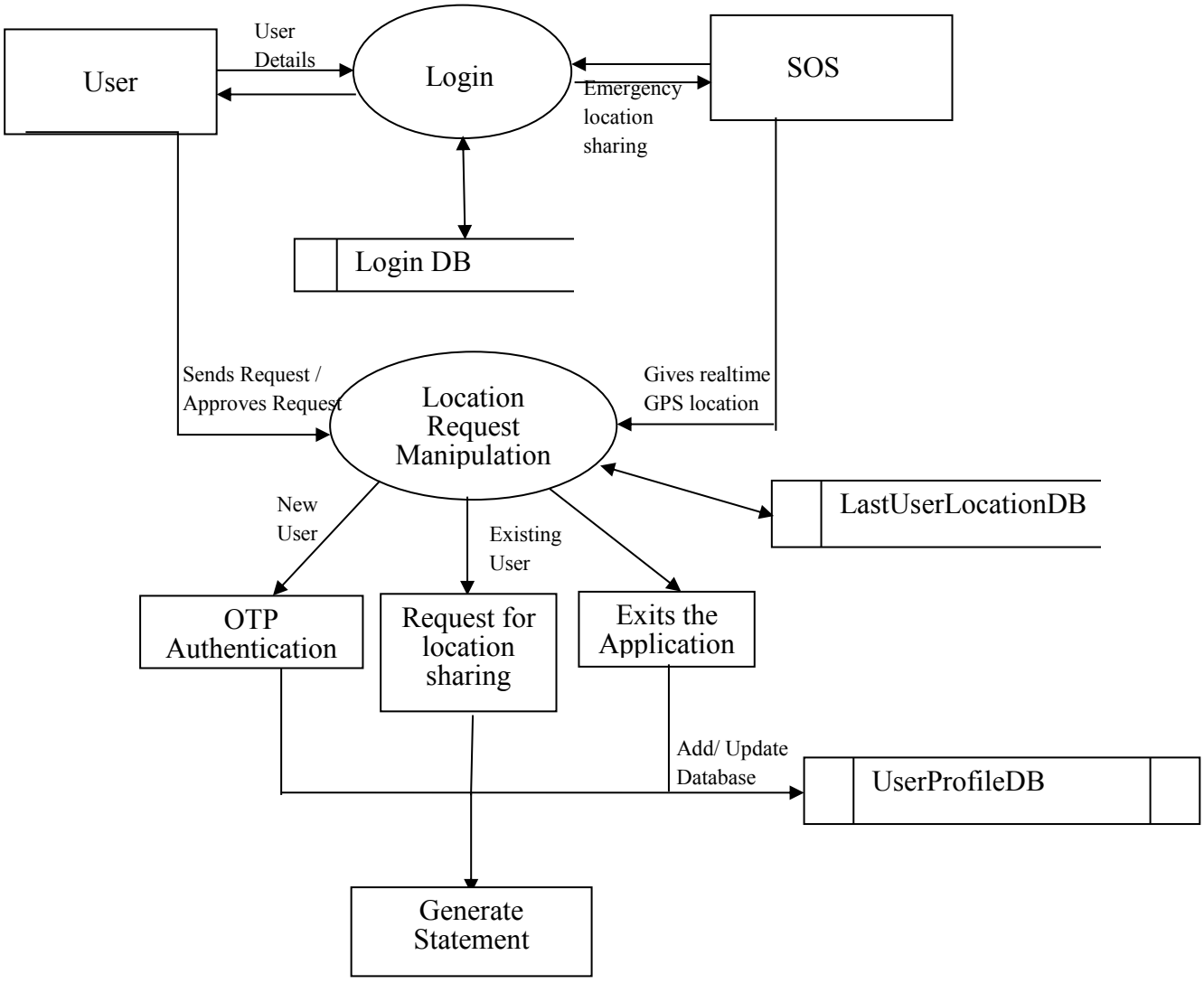
## Level - 0




## Level - 1



## Level – 2


User
User Details
Login
SOS
Emergency location sharing
Login DB
Sends Request / Approves Request
Location Request Manipulation
Gives realtime GPS location
LastUserLocationDB
New User
Existing User
OTP Authentication
Request for location sharing
Exits the Application
Add/ Update Database
UserProfileDB
Generate Statement

# 5 Change Management Process

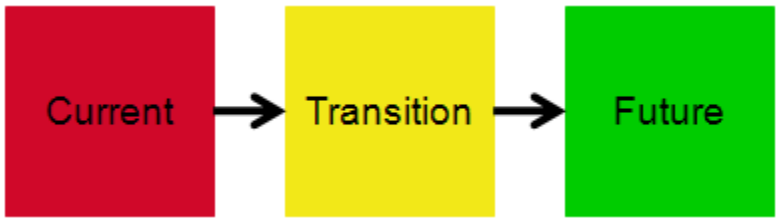
Change occurs as a process, not as an event. Organizational change does not happen instantaneously because there was an announcement, a kick-off meeting or even a go-live date. Individuals do not change simply because they received an email or attended a training program. When we experience change, we move from what we had known and done, through a period of transition to arrive at a desired new way of behaving and doing our job.

Although it is the last of the seven principles of change management presented, treating change as a process is a central component of successful change and successful change management. By breaking change down into distinct phases, you can better customize and tailor your approach to ensure individuals successfully adopt the change to how they work.

## 5.1 Understanding change as a process

It is easy to see changes in nature occurring as a process. Whether it is a caterpillar turning into a butterfly, or winter shifting into spring, we can easily appreciate the process of change. But when we begin changing our organization with projects and initiatives, we often forget the fact that change does not happen instantaneously.

The easiest, most basic approach to understanding change as a process is to break change down into distinct, understandable elements. The three states of change provide a powerful framework: the Current State, the Transition State and the Future State.




- **The Current State** - The Current State is how things are done today. It is the collection of processes, behaviors, tools, technologies, organizational structures and job roles that constitute how work is done. The Current State defines who we are. It may not be working great, but it is familiar and comfortable because we know what to expect. The Current State is where we have been successful and where we know how we will be measured and evaluated. Above all else, the Current State is known.

- **The Transition State** - The Transition State is messy and disorganized. It is unpredictable and constantly in flux. The Transition State is often emotionally charged - with emotions ranging from despair to anxiety to anger to fear to relief. During the Transition State, productivity predictably declines. The Transition State requires us to accept new perspectives and learn new ways of behaving, while still keeping up our day-to-day efforts. The Transition State is challenging.
- **The Future State** - The Future State is where we are trying to get to. It is often not fully defined, and can actually shift while we are trudging through the Transition State. The Future State is supposed to be better than the Current State in terms of performance. The Future State can often be worrisome. The Future State may not match our personal and professional goals, and there is a chance that we may not be successful in the Future State. Above all else, the Future State is unknown.

# A Appendices

## A.1 APPENDIX 1

| | |
|---|---|
| Core Java™ 2 Volume I – Fundamentals 7th Edition | Cay S. Hortsman |
| Pearson Education – Sun Microsystems | Gary Cornell |
| Core Java™ 2 Volume II – Advanced | Cay S. |

## A.2 APPENDIX 2

| | |
|---|---|
| O'Reilly – SPD | Elisabeth Freeman |
| The Book of JavaScript 2nd Edition | thauSPD |
| Effective Java – Programming Language Guide | Joshua Bloch |
| Pearson Education – Sun Microsystems | George Reese |
| JBoss – A Developers Notebook | Norman Richards |