+

**Software requirement specification**

**for**



**LocAdoc**

*A location based document locking application*

Version 0.1

11th August 2016

**Prepared by: Abhi Jay Krishnan**

**Kim Hyeocheol**

**Rivaldo Erawan**

**Durrah Afshan**

# Table of Contents

[Table of Contents i](#_Toc490226207)

[Revision History iii](#_Toc490226208)

[1. Introduction 1](#_Toc490226209)

[1.1 Purpose 1](#_Toc490226210)

[1.2 Document Conventions 1](#_Toc490226211)

[1.3 Intended Audience and Reading Suggestions 1](#_Toc490226212)

[1.4 Project Scope 2](#_Toc490226213)

[1.5 References 2](#_Toc490226214)

[2. Overall Description 3](#_Toc490226215)

[2.1 Product Perspective 3](#_Toc490226216)

[2.2 Product Features 3](#_Toc490226217)

[2.3 User Classes and Characteristics 4](#_Toc490226218)

[Physical Actors 4](#_Toc490226219)

[System Actors: 4](#_Toc490226220)

[2.4 Operating Environment 4](#_Toc490226221)

[2.5 Design and Implementation Constraints 4](#_Toc490226222)

[2.6 User Documentation 4](#_Toc490226223)

[2.7 Assumptions and Dependencies 4](#_Toc490226224)

[3. System Features 5](#_Toc490226225)

[Main Use Case Diagram 5](#_Toc490226226)

[Use Case 1.0: User Registration 6](#_Toc490226227)

[Use Case 1.1: Create User 7](#_Toc490226228)

[4. External Interface Requirements 8](#_Toc490226229)

[4.1 User Interfaces 8](#_Toc490226230)

[4.2 Hardware Interfaces 8](#_Toc490226231)

[4.3 Software Interfaces 8](#_Toc490226232)

[4.4 Communications Interfaces 8](#_Toc490226233)

[5. Non-functional Requirements 9](#_Toc490226234)

[5.1 Performance Requirements 9](#_Toc490226235)

[5.2 Safety Requirements 9](#_Toc490226236)

[5.3 Security Requirements 10](#_Toc490226237)

[5.4 Software Quality Attributes 10](#_Toc490226238)

[Appendix A: Glossary 11](#_Toc490226239)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 1. Introduction

## 1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to specify the requirements and to give detailed description of the functionalities of the locAdoc a location based document locking application. This document will cover each features of the system, software requirements, and the targeted audience.

## 1.2 Document Conventions

Main Section Titles

* Font : Calibri Light
* Face : Bold
* Size : 16

Sub Section Titles

* Font : Calibri Light
* Face : Bold
* Size : 14

Other Text Explanations

* Font : Calibri Body
* Face : Normal
* Size : 12

## 1.3 Intended Audience and Reading Suggestions

This document is intended for all individuals taking part in the locAdoc application development, such as developers, project managers, users, supervisor and testers.

Readers who are interested in the overview of the project should concentrate on Part 1 which will give an introduction of the project. Part 2 of the document describes a brief overview of the project in each aspect.

Readers who want to understand the project as a whole should focus on Part 3 which provides the features of the project in detail. Part 4 will give the visualization of the project as well as the hardware, software, and communication requirements of the project.

Readers interested in the nonfunctional requirements of the project should read Part 5, which gives information about the performance, safety, security, and other attributes.

## 1.4 Project Scope

This project aim to provide user a way to store confidential documents in mobile devices and access it only in the area he/she find it is safe. By including two factor protections, one being password (what the user knows) and second being the location (where the user is currently), we will be able to provide a better solution compared to the applications currently in the market (based on market survey).

These are few ways a document in a mobile device may be compromised: -

* The documents stored in mobile device may end up in the wrong hands if the device itself is stolen.
* The user may lend the device to someone who intern may wish to gain access to these documents.
* The documents may be accessed remotely by penetrating device through network.

Our solution aim to provide a secure vault for document storage so the it does not get into wrong hands even if the device is compromised. The solution also provide a secure backup cloud storage with double layer encryption one by the app itself and one by Amazon server.

## 1.5 References

SRS Template by Karl E. Wiegers

http://users.encs.concordia.ca/~eshihab/teaching/slides/srs\_template\_sep14.pdf

# 2. Overall Description

## 2.1 Product Perspective

LocAdoc (CCFBS) is a new, self-contained application running on android platform written in Java. It is intended in providing the user a safe place to store his pdf files and view it at an area he feels secured. The system uses a client cloud architecture where the client is the application running on a mobile device and it makes use of cloud services provided by amazon web services.

## 2.2 Product Features

The features of this system are:

***PDF Viewer***

A PDF viewer for user to view his documents in the vault. The pdf viewer will only be accessible after the user has been authenticated and if the user is within the radius of the location stored in the database. The pdf viewer will close when the user moves out of this zone. The file that the user wishes to see will be the only one that will be decrypted. The rest will remain as cipher text even when the user is in authorised area. This pdf viewer will help the user to be more productive by having the ability to access sensitive document while moving within the secure location.

***Deleting files***

The user has the option to delete the files that are not needed and these files will also be deleted from the backup.

***Setting preferred locational radius***

Once the user adds a new file he can set the radius he wishes with small radius being more secure and larger radius being more convenient. The files will be grouped based on the location and the user can choose the area if there is an overlap.

***Less clustered interface***

The user will be only able to view the files that was saved to a location making file accessing, pleasant and less tedious.

***Import files***

The user will be able to import a new file from the local file directory and secure it through encryption. The original file can be deleted to prevent adversary from viewing it.

***Secure cloud storage***

The data will be safely stored in the central database with additional layer of encryption by the cloud infrastructure provider.

These are the key features that will be included in the application. Further enhancements such as support for more file types will be added if these basic requirements are met.

## 2.3 User Classes and Characteristics

Physical Actors**:**

* **Mobile users:** The user who uses the system and make use of the services provided by the application.

### System Actors:

* **DynamoDB:** A NoSQL database service provided by Amazon web services(AWS) and this service will be used to create a central application database.
* **AWS Cognito:** A user password authentication service provided by Amazon and will be used to authenticate user based on a central user pool.
* **AWS S3:** A central file storage facility provided by the AWS will be used to backup user data.

## 2.4 Operating Environment

This system only operates in Android Operating System.

## 2.5 Design and Implementation Constraints

The main constraint of this program is the support for several file formats, as each file format will require dedicated viewer. For now, we are sticking to PDF format and we will consider other formats for future releases.

## 2.6 User Documentation

The user can use the help menu in the system to understand the interfaces more.

## 2.7 Assumptions and Dependencies

* Each member must have a UserID and password.
* The system may run online (via the Internet).
* Facilities and Members are fully observed by the Club Manager.

# 3. System Features

## Main Use Case Diagram



## Use Case 1.0: Sign up

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 1.0 |
| **Name** | Sign up |
| **Description** | To allow new user to create a new account after he download the application for the first time. |
| **Actor(s)** | User who is interacting with the application, Signup activity, AWS Cognito, AWS DynamoDB, AWS S3. |
| **Precondition** | A user has installed the application for the first time and wish to create a new account. |
| **Main Scenario** | Step 1: System prints out sign up form with fields asking for users particulars.  Step 2: User submits the form and the system does form validation.  Alternate Step 2: If the data is invalid, an error message is displayed and the user is sent back to the form to reenter the data correctly.  Step 3: The user details are encrypted using the newly entered password.  Step 4: The system updates the AWS Cognito server.  Step 5: The system updates the AWS Dynamo DB.  Step 6: A new secure folder is created  Step 7: A backup is created in the AWS S3.  Step 8: The user is directed to the login screen. |

## Use Case 2.0: Login

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 2.0 |
| **Name** | Login |
| **Description** | To allow existing user to enter and use the if the user have signed up or create and account in the system |
| **Actor(s)** | User who is interacting with the application, System, AWS Cognito, AWS Dynamo DB, SQLite |
| **Precondition** | A user has an active account in the system (sign up) and wish to use the system |
| **Main Scenario** | Step 1: Activity prints out sign in form with fields asking for user’s account and password.  Step 2: User fills in the form and submits it, then the activity does form validation.  Step 3: If the data is valid the activity hands over the encrypted details to the AWS Cognito.  Alternate Step 3: If the data is invalid, an error message is displayed and the user is sent back to the form to reenter the particular correctly.  Step 4: The activity receive a success reply from AWS Cognito, the user will enter the system.  Alternate 1 of Step 4: The activity receive a fail reply AWS Cognito and the user is sent back to the form to reenter the particular correctly.  Alternate 2 of Step 4: The activity consecutively receive a fail reply from AWS Cognito 3 times, the system will be locked for a certain period of time. After that every consecutive fail will lock the system, and the time will increase as the number of failed consecutive log in increases. It resets when a correct particular is entered  Step 5: Import users data from AWS Dynamo DB to local SQLite database |

## Use Case 2.1: MAC address verification

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 2.1 |
| **Name** | MAC address verification |
| **Description** | This is a process runs in the background to tell user if the MAC address has changed (mobile device). It lets the user to change password if the MAC address has changed |
| **Actor(s)** | User who is interacting with the application, Activity, AWS Dynamo DB, AWS S3,Device. |
| **Precondition** | The user has successfully logged in to the system |
| **Main Scenario** | Step 1: Activity compare the MAC address in the AWS Dynamo DB with the MAC address of the device used.  Step 2: If the MAC address is different, the activity will display the notification message and give the user an option to change the password  Alternate Step 2: If the MAC address is same, the user will enter the system  Step 3: If the user decides to change password, a form will be printed  Alternate Step 3: If the user decides to not change the password, go to Step 6  Step 4: User fills in the form and submits it, then the activity does form validation.  Step 5: If the data is valid the activity will update the new password to the AWS Dynamo DB  Alternate Step 5: If the data is invalid, an error message is displayed and the user is sent back to the form to reenter the new password correctly.  Step 6: Enter the system |

## Use Case 2.2 Password Recovery

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 2.2 |
| **Name** | Password Recovery |
| **Description** | IF user forget their own password to login, the mobile app available to recover user’s password by connecting with Amazon Identity Provider and Amazon Web Service Cognito. |
| **Actor(s)** | User who is interacting with the application, Amazon Identity Provider, Amazon Web Service Cognito, STS(Security Token Service) and DynamoDB and Login activity |
| **Precondition** | User has the mobile application in mobile device and account(User ID + password) in the Mobile App. And the account should have the verified either user Email or Contact Number to authentication in recovery step. |
| **Main Scenario** | Step 1: Activity display Login Screen  Step 2: User select the Forgot Password in Login Screen to recover user’s password.  Step 3: The activity requests the user for password recovery method (Email/Short Message Service).  Step 3: User choose the Recovery method.  Step 5: (Backend) The activity and AWS Cognito exchange the identities (Authenticated identities or Guest identities).  Step 6: After Exchange, AWS STS (Security Token Service) send the temporary token to End user’s email or phone (SMS) based on the user’s choice in Step4.  Step 7: IF user input valid token, THEN end user can login to access the app. And then the app will display the reset password form to change user’s password.  ALTERNATE Step 7: IF user input invalid token, THEN the app will display error message. And then resend the temporary token to user again.  Step 8: The login activity load change password activity.  Step 9: The user enters new password and the system updates both local and central AWS DynamoDB database. |

## Use Case 2.3: Data Recovery

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 2.3 |
| **Name** | Data Recovery |
| **Description** | To allow user to retrieve backed up data from AWS S3 when he change device or removed the app |
| **Actor(s)** | User who is interacting with the application, System, AWS S3 |
| **Precondition** | The user has successfully logged in to the system |
| **Main Scenario** | Step 1: Activity check if all the files inside database exist in the local storage  Step 2: If it doesn’t, the activity will retrieve back up data from ASW S3 and show the appropriate message  Alternate Step 2: If it does, enter the system  Step 3: Enter the system |

## Use Case 3.0: Import documents

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 3.0 |
| **Name** | Import document |
| **Description** | This functionality makes sure that the data imported to the sever application is stored in a secured manner. |
| **Actor(s)** | User who is interacting with the application, Import Document Activity, GPS(System), AWS DynamoDB, Local SQLite database. |
| **Precondition** | The user wishes to secure a new document. |
| **Main Scenario** | Step 1: On clicking the import file option the user is prompted brows and enter the location of the file in the file system.  Step 2: The activity records the current location of the user (from GPS system).  Step 3: The activity searches the database and loads areas that where previous created in same location. The user is prompted to choose the radius where he wishes to access the file. The system groups the new file along with the order files in same area.  Alternate to step 3: The user may choose to create a new area for the current file. If he wishes to create a new area he will be promoted to set the radius of his choice.  Step 4: The activity will update the database and the file will be encrypted using the longitudinal and latitudinal value where the file was imported.  Step 5: The activity accesses the user’s password and concatenates it with the current longitude and latitude received from phones GPS (System).  Step 6: A key is produced by hashing password digest produced in step 2 (Hash(pwd|Locationaldata)).  Step 7: The activity creates a new file name for the file and this name is mapped with the original name in the database name.  Step 8: All the above generated data is populated to local database (SQLite and the central database (DynamoDb) in the cloud.  Step 9: The file is encrypted using AES 256 and the key generated in Step 3. |

## Use Case 4.0: View Document

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 4.0 |
| **Name** | View Document |
| **Description** | To allow user to view its encrypted documents |
| **Actor(s)** | User who is interacting with the application, System, SQLite DB |
| **Precondition** | The user has successfully logged in to the system and is within the area of the document |
| **Main Scenario** | Step 1: Activity show the list of documents available in the user’s current position  Step 2: User select the desired document  Step 3: The activity retrieves the password used to encrypt this document as well as the location of data from local SQLite DB  Step 4: The activity produces the corresponding key by hashing password digest (Hash(pwd|Locationaldata))  Step 5: The activity decrypts the document with AES 256 using password digest computed in Step 4 as its key  Step 6: The activity shows the document to the user  Step 7: When the user is viewing a file, the activity checks his location every five seconds.  Alternate Step 7: When the user moves out of the secure location the viewer will be closed and the file will be encrypted. A warning would be displayed if the user approaches the boundary.  Step 8: When the user exit the viewer, the document is encrypted back with AES 256 using password digest computed in Step 4 as its key |

## Use Case 5.0: Data Back up

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 5.0 |
| **Name** | Data Back up |
| **Description** | To allow user to back up its encrypted documents to AWS S3 |
| **Actor(s)** | User who is interacting with the application, Data backup activity, AWS S3 |
| **Precondition** | The user has successfully logged in to the system |
| **Main Scenario** | Step 1: User select to back up its data  Step 2: Activity check the total size of the backup data (Check limits, by default 500MB).  Step 3: If it is within the limit, all the encrypted data is uploaded to AWS S3  Alternate Step 3: If it exceeds the limit, activity will cancel the process and show the appropriate message |

## Use Case 6.0 Delete Document

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 6.0 |
| **Name** | Delete Document |
| **Description** | This Function allow user to delete the document file in device and database in Server. Deleting file in Data Backup File is optional. |
| **Actor(s)** | User who is interacting with the application, AWS DynamoDB |
| **Precondition** | A user has installed the application. Also user has an account in the app and some files in device, database and backup file. Before execute this function, user must already login to the app. |
| **Main Scenario** | Step 1: User select the Delete documents option in Menu Display  Step 2: The Mobile app request to AWS server with sending the user’s location. THEN the AWS server searches the documents, matching with the user’s location, in database.  Step 3: AWS server send the documents list. Then the app will show the document List depends on the user’s current location.  Step 4: When there are documents, The app ask to user to select files for deleting.  Step 5: User select the documents and execute the delete.  Step 6: App will ask to user whether delete the documents in backup file or not.  Step 7: IF user selects YES to delete documents in backup file, THEN App ask to server to delete file in database only.  ALTERNATE Step 7: IF user selects NO to delete documents in database only, THEN App ask to server to delete file in database only.  Step 8: AWS server executes to delete the selected files depends on the user’s request AND THEN reply the updated list to the app.  Step 9: App show the updated documents list to the user.  Step 10: User go back to Main Menu in the app.  ALTERNATE 10: IF there are files which user wants to delete, then go back to Step 4. |

## Use Case 7.0: File Recovery

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 7.0 |
| **Name** | File Recovery |
| **Description** | To allow user to retrieve backed up data from AWS S3 when the user accidentally removed a file |
| **Actor(s)** | User who is interacting with the application, System, AWS S3 |
| **Precondition** | The user has successfully logged in to the system |
| **Main Scenario** | Step 1: User select to recover a file.  Step 2: Activity show the list of documents that are available to recover in the user’s current position  Step 3: User select the desired file  Step 4: Activity retrieve backed up file from AWS S3 and saves it to the local storage.  Step 5: The user is notified. |

## Use Case 8.0 Delete account

|  |  |
| --- | --- |
| **Use Case Textual Description** | |
| **UC-ID** | 8.0 |
| **Name** | Delete account |
| **Description** | This Function allow user to delete user’s account in the Application Server. |
| **Actor(s)** | User who is interacting with the application, AWS DynamoDB |
| **Precondition** | A user has installed the application. Also user has an account in the application. user locates in valid area to delete account. Before execute this function, user must already login to the app.  The Location Type will be divided as Normal Type or Admin Type.  Default Mode will be Normal Type in location, which allow user to access and modify account data in whole area. (Whole areas are set as Normal Area)  If user selects the more secure mode in the app, then user can access and modify the user account only in Admin Type of location. (There are some admin area and normal area.) |
| **Main Scenario** | Step 1: User select the Delete account option in Menu Display  Step 2: The Mobile app request to AWS server with sending the user’s current location.  **METHOD1.** Normal Secure Mode(Only Normal Area Type)  Step 3.1: Then AWS server compare user’s current location area with user defined area in database  Step 4.1: IF user locate in valid Normal Area, then the AWS server send message to App whether user will delete the account  ALTERNATE Step 4.1: If it is not valid area, then the server send reject message to app and app display reject message.  **METHOD2**. More Secure Mode(Using Both Normal Area Type and Admin Area Type)  Step 3.2: Then AWS server check user’s current location area whether it is Admin Area Type or not.  Step 4.2: If it is Valid Admin Area Type, then the AWS server send message to App whether user will delete the account,  ALTERNATE Step 4.2: If it is not valid area or not Admin Area but Normal Area Type, then the server send reject message to app and app display reject message.  Step 5: IF user Select yes to delete account and AWS server delete the account with its document in database and backup file.  ALTERNATE Step5: IF user select no, then terminate the process and go back to main menu.  Step 6: After Deleting Account, The app will logout and display login format. |

# 4. External Interface Requirements

## 4.1 Hardware Interfaces

The Hardware mainly consist of the mobile phone with android operating system and a central Amazon server. The Amazon server provides all the necessary infrastructure along with maintenance to do deploy the cloud storage service and a central database. All the mobile devises will make connection with this central server.

## 4.2 Software Interfaces

The client-side software will be installed from the android store and which then will retrieve all the necessary user information from Amazon DynamoDB. The Amazon service will scale as the number of users using the service increase.

## 4.3 Communications Interfaces

The communication interface is handled by AWS mobile API. All information passed to the API will be encrypted by the application. The communication is conducted over internet.

# 5. Non-functional Requirements

## 5.1 Performance Requirements

A temporary SQLite database will be created to store the user credentials locally to costs (time) that is incurred through constant communication with the central database. This database will be dropped when the user logout. This will boost the performance as the encryption and decryption processes will be heavily depend on this data. The performance is key requirement as mobile devises are small and don’t have computational power of personal computers.

## 5.2 Safety Requirements

One of the key non-functional requirement is to keep the user data safe. Following are some of the identified Safety requirements: -

***Dealing with stolen device***

The user can migrate to a new device using the backup data and change password so that the data in the old phone can never be decrypted as the sever will never authenticate the user even if he knows the old password.

***Dealing with missing folder***

The user will be promoted to download the backup folder and will be able to access files as usual or the user can choose to create a new one.

***Migration***

The user may wish to move to a new device, and the app allows user to download the backup file from the cloud and continue using as usual. The user credentials are stored in a central database to ease migration.

***Backup***

The user can back up his data to cloud (AWS S3) to retrieve it later when the disaster strikes or for migration. The system will track the changes and only save the newly added file. The data stored in cloud is send over after encryption.

***Changing Password***

The user after logging into the app can change the password by providing the old password. The password will be saved to the central database. The password forms a part of the key used to secure files and tables in the database. Once the password is changed the database will be secured using new password when the user logout.

***Password recovery***

If the user forgets the password he will receive a password recovery code through his email or notification. Upon entering the code, he will be redirected to the changing password procedure.

***Protection against GPS spoofing***

The location will be checked regularly (every 10 sec) and will be compared with the previous location, if there is a significant difference in the coordinates (100m) the app will encrypt the files and logout.

## 5.3 Security Requirements

Since the main aspect of this application is to keep files secure, this requirement is to non-trivial. Files should be encrypted using secure cryptographic algorithm such as Advanced Encryption standard (AES256). The file should not have the same name as the original file after encryption, the original name will be stored in the database along with the new name. The password should be hashed using strong hash functions such as the SHA256. The central database will have an additional layer of encryption provided by the cloud service provider (Amazon Web Service).

As for SQL injection, the query is performed by AWS Mobile API hence the adversary will not be able to inject malicious code to the server. All data that is sent over is not executed and will be stored in encrypted format.

## 5.4 Software Quality Attributes

**Availability:** Checking that the system always has something to function and always pop up error messages in case of component failures. In that case the error messages appear when something goes wrong to prevail availability problems.

**Usability:** Checking that the system is easy to handle and navigates in the most expected way with no delays. In that case the system program reacts accordingly and transverses quickly between its states.

**Functionality:** Checking that the system provides the right tools to perform task mentioned in the functional requirement section. Also testing these functionalities run smoothly and providing simple learnable interface.

# **Appendix A: Glossary**

**Actors**: These are the stakeholders of the system. It may be a person, a group of people or an external system that may either support or use the main system.

**Architecture**: It is the highest level structure of the software system. The whole system is developed based on the software architecture.

**Client**: These are the systems that interact with the user and transfer data to a central server.

**Club Member**: Members of the country club those who will be using the system to make bookings.

**Club Manager**: The manager of the club who uses the system to make changes to the background data of the system on which the booking depends on.

**C++**: It is a middle-level programming language developed by Bjarne Stroustrup starting in 1979 at Bell Labs.

**Database system**: This is the place where the data is stored. We are currently using a text file to store the data on server side but it can be upgraded to a full-fledged database system.

**Linux**: Is a Unix-like operating system developed in bell labs. It is free and open-source.

**Ranking**: Refers to the different membership rights in the club. Different rank members have different priorities, different access rights and different discount rates.

**Rational Unified process (RUP):** Quoted from <https://en.wikipedia.org/wiki/Rational_Unified_Process> “The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation, a division of IBM since 2003.[1] RUP is not a single concrete prescriptive process, but rather an adaptable process framework, intended to be tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs. RUP is a specific implementation of the unified process.” RUP has following phases:-

* **Inception**: Collecting ideas.
* **Elaboration**: Documenting and designing system based on the collected ideas.
* **Construction**: Developing the design into a programme and testing the system.
* **Transition**: Deploying the system.

**Server**: They are the back end subsystem which contains the entity class and the data files. They manage the flow of data to the client. They also fulfill the requests from multiple clients at the same time.

**System**: A system is a set of procedures according to which a problem is solved. A system has various subsystems which perform sub tasks to achieve a common goal.

**Subsystem**: A subsystem is a system containing various use cases which perform sub tasks to achieve common goal. Subsystem fulfill a major functionality of the entire system.

**TCP/IP—Transfer Control Protocol/Internet Protocol**:  Internet protocol suite is the computer networking model and a set of communications protocols used on the Internet and similar computer networks. TCP/IP provides end-to-end connectivity specifying how data should be packetized, addressed, transmitted, routed and received at the destination.

**Use-case**: It is the key functionality of the system.