# Travel in Good Health Management

Software Requirements Specification (SRS) Document

M Tharun (14CS10056)

S V Dilip Kumar (14CS30030)

●

Software Engineering
Assignment - 2016

# Table of Contents

# 1. Introduction

## 1.1 Purpose

This document specifies the detailed description of the requirements for the "Travel in Good Health Management" (TGHM) software. This document will illustrate the purpose and the complete declaration for the development of the system. This document is proposed to the client for his approval and as a reference for the development of the first version of the product by the developing team. It helps in understanding how the client or the customer wishes to benefit from the software, so that the final product has all functionalities that the client needs. It primarily clears the ambiguities in the mind of the developing team before the process of system development commences. It will also explain the interface and interaction with other external applications and will also declare the assumptions made and the constraints of the system.

## 1.2 Project Scope

The software "TGHM" is that creates a bridge between passengers and restaurants such that the passenger could have healthy food while travelling. Restaurants and food chains can register with management as food suppliers along with their agents who are crucial for the accepting orders and delivering the food to the customer. The system links the customer with the agents of the restaurants along the route of the train and suggests optimal restaurants in the vicinity. The restaurant provides its menu to the management which can be referred by the customer while placing an order. The order contains details about time of delivery, restaurant and food item. The customer may not provide all the details in which case the system needs to suggest options available. The system informs the agents with respect to their success rate in delivering food and the total number of orders they have handled. The management has the ability to cancel registration requests sent by the restaurants and food chains by validating the details of the food provider and the authentication certificate produced. The system also sends a confirmation receipt once the order has been successfully placed. This software depends on a well maintained database management System. The GUI of the software is made in Java using NetBeans and user-friendliness is made top priority.

## 1.3 Audience and Reading Suggestion

The common audience of this document is The Management of TGHM, and the Software Developer. They await response from client sides regarding refinement in the process suggested to be deployed.

## 1.4 Glossary

| | |
|---|---|
| TGHM | Travel in Good Health Management |
| Customer | Person who orders food during or before their train journey. |
| Management | The body in charge for facilitating the conversation between the restaurant/food chain and the customer. It holds record of the train routes and the corresponding to which its services are available and also the details about the restaurants (location/contact no/authentication certificate) and its agent details such as the stations he could reach. It optimizes the suitable restaurants for the placed order and suggests the customer details. |
| Agent | The agency which deals with the delivery of the food and the one which is responsible for accepting or rejecting the order placed by the customer. He delivers the food to the station in the least wait time possible. |
| Restaurant/Food Chain / Food Provider | It is the one that provide the food to the agent for delivery and decides the delivery time for preparation of the order. |
| Order | The list of food items ordered by the customer |
| | |

### 1.5 References

IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirement Specifications. IEEE Computer Society, 1998.

# 2. Overall Description

## 2.1 Product Perspective

The software “TGHM” takes advantage of the digital revolution to cater the nutritional needs of the customers while travelling in train which aims on taking over pantry car system which provides undesirable food to the passengers. It is efficient in the sense that it links the customers with trusted food providers who could guarantee quality food during the journey .The pantry car system is a good way to eat ready-made food but could not a substitute in case of long run. TGHM system seems to be efficient in the way that it could track almost all registered restaurants along the train route so that trusted restaurants near each halting station can be detected and suggested to the customer which could provide food for a nominal price. This is indeed efficient in the sense that the customer no longer needs to worry about his hygiene before he ventures on a journey as he has a wide variety of food choices to choose from and also has the flexibility to decide what timings could be convenient for receiving the food. Also it doesn’t restrict the customer from imposing his choices

## 2.2 Product Features

The customer can initially place order with restaurant, time of delivery, food item attributes and can modify the placed order if rejected by an agent. The management can create orders and forward it to the most suitable agent depending on the order provided. It can add new food providers (along with authentication certificate, location, and contact number) and agents (stations he would be able to reach) as well as new train routes and trains and also remove existing ones. The agent can accept orders and deliver the order to the customer. The management notifies the customer once an order has been rejected with an apology and then request for any changes to the existing order. The food provider can decide the time taken for delivery of the items to the customer.

## 2.3 User Classes and Characteristics

The management of system is the user who’ll use this product. He has access to all the features of the product. The management has to create an order when required when placed and forward it to an agent repeatedly until one accepts. He has to add new restaurants and agents to the system as well as remove them if necessary. And he also has to apologize when an order has been cancelled. From these tasks the management has right to edit the database of the system and hence the major user.

The different use case diagrams are:

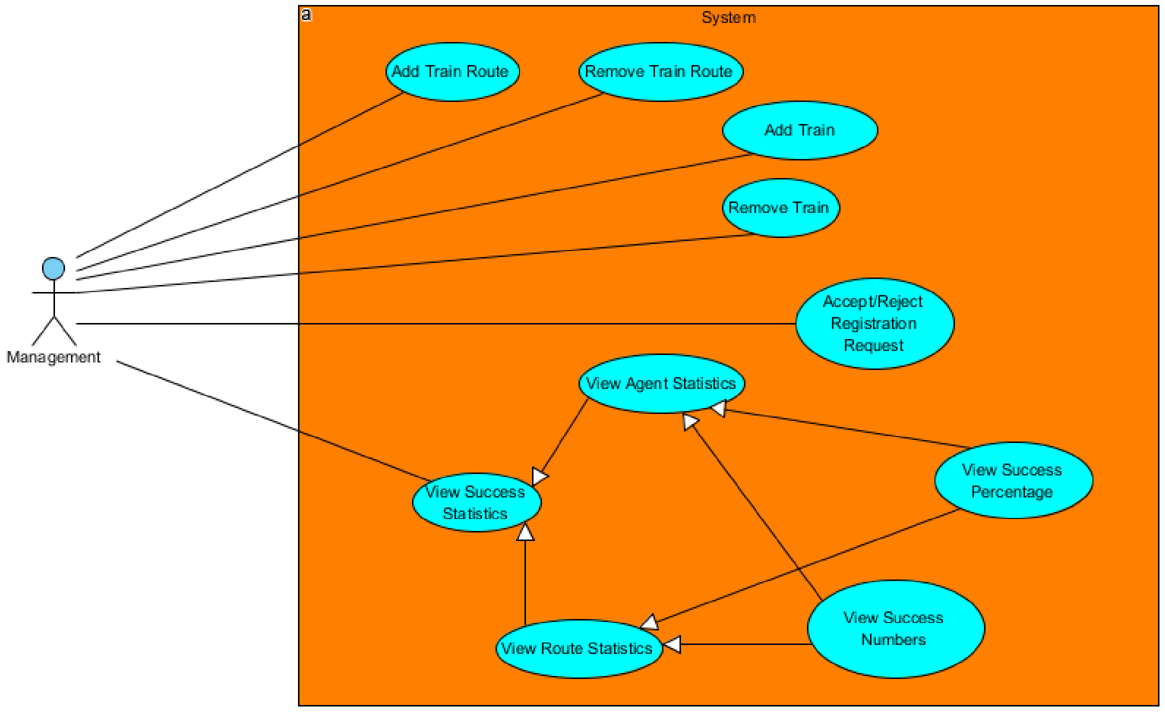
- Management – Authority to add/remove train route, add/remove train, approve/reject registration request while a food supplier registers with the management, and view success statistics of agent or a particular train route before suggesting restaurants to the customer.

- Customer – Place order with the management, receive food from the agent, modify/ cancel order once rejected, pay bill before delivery of food, and receive receipt once the order has been placed.

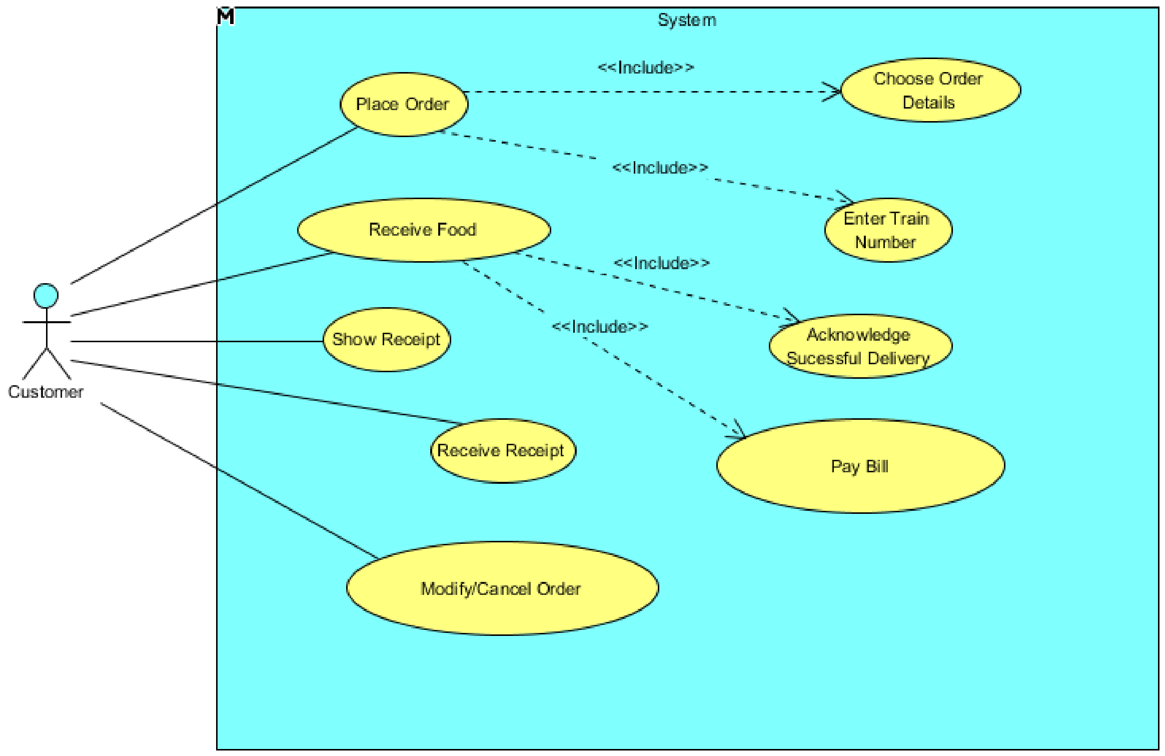
- Agent – Provide details once the respective food provider has registered with the management, accept/ reject order forwarded by the management.

- Food Supplier – Registers with the management as Food Provider, enter the details of the menu items served, modify menu subsequently.

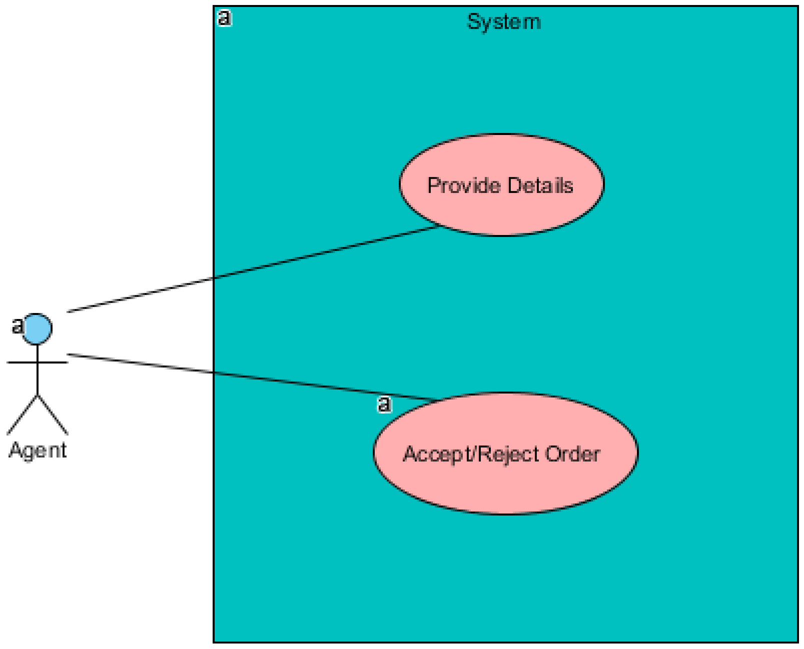
*Use case Diagram for user class Management:*



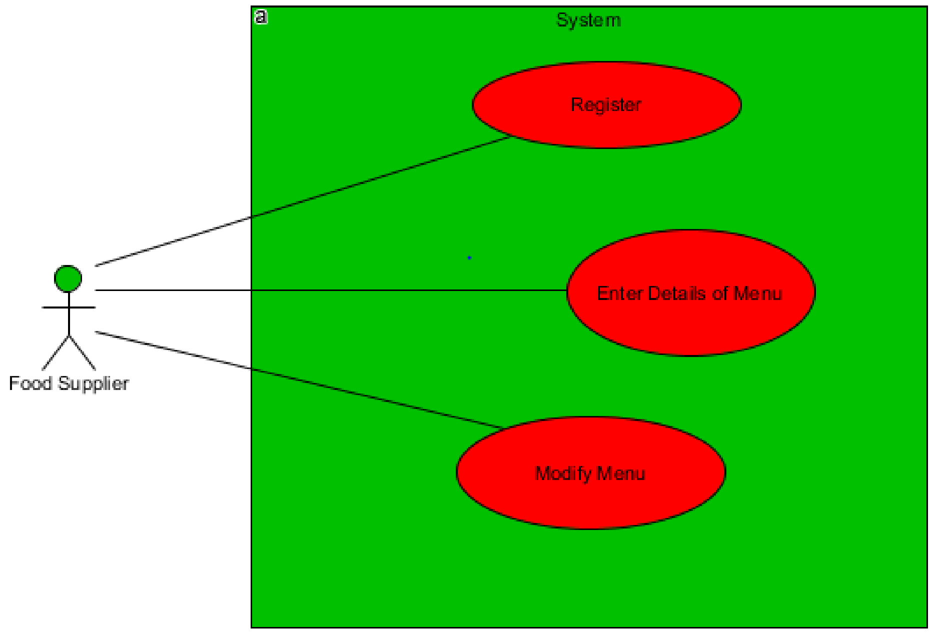
*Use case Diagram for user class Customer:*



*Use case Diagram for user class Agent:*



*Use case Diagram for user class Food Supplier:*



## 2.4 Operating Environment

It is operative in any operating system. The only requirement is that the environment should have the current version of the jdk (java development kit) (jdk 1.7 or above). As the programming is done in Java (NetBeans IDE), so there is a requirement of the java compiler (javac) pre-installed and working in the operating environment.

## 2.5 Design and Implementation Constraints

Connection is a constraint for the application. Since the application fetches data from the database (which will probably reside on a single centralized server), it is crucial that there is a connection for the application to function. The application will be constrained by the capacity of the database. The database may be forced to queue incoming requests and therefore increase the time it takes to fetch data.

## 2.6 User Documentation

The primary goal of the TGHM is to facilitate the process of availability of hygiene food during travel. Consequently, the application will be designed to be as simple to use as possible. Nonetheless, users may still require some supplementary information about each component of the TGHM. The application will contain a Help Menu which will provide a tutorial on how to use the software. It contains all necessary facts for understanding the operation of TGHM keeping in mind the ease of use of the software.

## 2.7 Assumptions and Dependencies

The following is a list of assumptions made about the TGHM that will have effect on the working of the software:

- The customer can provide partial details while ordering and the management produces suitable suggestion satisfying the choices given by the customer.
- The food provider has a single agent and an agent works for a single particular restaurant.
- The stations that the food provider can provide is decided by the agent of the food provider.
- After an agent places an order the customer has two choices he could either allow modification of his order or he could cancel the order. Note that the restaurant changes for the next suitable agent hence the order details which contains the restaurant name is obviously modified. Even though the customer doesn't explicitly modify it.
- The customer pays for his food after producing the receipt during the time of delivery.

The following is the list of dependencies:

- A proper network connection exists between the server (on which the medical test database resides) and the various travel management sub-systems which try to access the database.
- The server has a properly installed version of MySQL (used for the database).
- All the sub-systems using TGHM should have JDK 1.7 or above properly install led.

# 3. Functional Requirements

## 3.1 User Class – Management

### 3.1.1 Add Train Route

- Input:
  Details such as route name, route number are entered by the management.
- Output:
  A new train route object is added to the database.
- Process:
  The management checks for existing train routes if the new train route entered is not present, then it is added to the database.

### 3.1.2 Remove Train Route

- Input:
  The Train Route Object which needs to be removed is mentioned by the management.
- Output:
  The mentioned train route is deleted from the database.
- Process:
  The management checks for existing train routes if the mentioned train route is present, then it is deleted from the database.

### 3.1.3 Add Train

- Input:
  Details such as train name, train number are entered by the management.
- Output:
  A new train object is added to the database.
- Process:
  The management checks for existing trains if the new train entered is not present, then it is added to the database.

### 3.1.4 Remove Train

- Input:
  The Train Object which needs to be removed is mentioned by the management.
- Output:
  The mentioned train is deleted from the database.
- Process:
  The management checks for existing trains if the mentioned train route is present, then it is deleted from the database.

### 3.1.5 Approve Registration Request

- Input:
  A restaurant/food chain sends a request to the management asking for registration in TGHM system as a food provider.
- Output:
  Management approves or rejects the request sent in by the food provider.
- Process:
  The management checks the validity of the authentication certificate and details such as contact number, location of the registering food supplier. If the details seem legit the restaurant/food chain is added to the database of the system as a new food provider else the request is rejected.

### 3.1.6 View Success Statistics

- Input:
  The Train Route Object which needs to be removed is mentioned by the management.
- Output:
  The flow is directed to use cases 3.1.7 or 3.1.8

- Process: Depending upon what statistics the management wishes to view the control is directed towards success of a particular agent or the success of a particular train route.

### 3.1.7 View Agent Statistics

- Input:
  Details about the agent for whom the success statistics are to be viewed.
- Output:
  The flow is directed towards use case 3.1.9 or 3.1.10
- Process:
- Depending upon whether the management wishes to view the success percentage or the success numbers the flow is directed towards the corresponding use cases.

### 3.1.8 View Route Statistics

- Input:
  Details about the route for which the success statistics are to be viewed.
- Output:
  The flow is directed towards use case 3.1.9 or 3.1.10
- Process:
  Depending upon whether the management wishes to view the success percentage or the success numbers the flow is directed towards the corresponding use cases.
-

### 3.1.9 View Success Percentage

- Input:
  Use Cases 3.1.7 or 3.1.8
- Output:
  The percentage of success is displayed.
- Process:
  The percentage of success of the agent or the route whichever invokes this use case is displayed.

### 3.1.10 View Success Numbers

- Input:
  Use Cases 3.1.7 or 3.1.8
- Output:
  The number of success is displayed.
- Process:
  The number of success of the agent or the route whichever invokes this use case is displayed.

## 3.2 User Class – Customer

### 3.2.1 Place Order

- Input:
  Customer registers a new order with the management.
- Output:
  The flow is directed toward use cases 3.2.2 and 3.2.3.
- Process:
  The customer requests places an order with the management regarding the food items.

### 3.2.2 Choose Order Details

- Input:
  - Time of Delivery
  - Restaurant
  - Food Items
- Output:
  System updates the database with a corresponding order.
- Process:
  The system creates a new order with the corresponding details provided.

### 3.2.3 Enter Train Number

- Input:
  - Train Name
  - Train Number
- Output:
  System updates the database with the corresponding order.
- Process:
  The system updates the already existing order with the train attributes.

### 3.2.4 Receive Order

- Input:
  The customer produces the receipt sent by the management to the agent and pays the bill.
- Output:
  The agent delivers the food to the customer and the flow is directed to the use case 3.2.5.
- Process:
  The agent validates the receipt produced and  collects the cost of food items and then delivers the food items.

### 3.2.5 Acknowledge Successful Delivery

- Input:
- Invoked by Use Case 3.2.4.
- Output:
  The customer verifies successful delivery to the management.

- Process:
  The customer checks for the food items whether they confirm with the order and indicates successful delivery.

### 3.2.6 Modify/Cancel Order

- Input:
  New order details which may vary in fields corresponding to the previous order.
- Output:
  System updates the database with a modified order or could cancel his order.
- Process:
  Once an agent has rejected the order the system modifies the order and the customer has the option to modify his previous order.

### 3.2.7 Pay Bill

- Input:
  Foods items delivered by the agent.
- Output:
  The agent receives the bill amount and passes the control to Use Case 3.2.8.
- Process:
  The customer checks the food items and pays the bill amount claimed by the agent.

### 3.2.8 Approve Successful Delivery

- Input:
  Invoked by Use Case 3.2.7.
- Output:
  The agent updates database incrementing both the number of successful deliveries and the percentage of success.
- Process:
  The customer approves that the agent has satisfactorily delivered the right food items in the order.

## 3.3 User Class – Agent

### 3.3.1 Provide Details

- Input:
  - Stations able to reach by the agent.
- Output:
  The details entered by the agent are stored along with the details of the restaurant in the database.
- Process:
  Once the restaurant has provided valid details, the agent is needed to provide the details next which are stored in the database.

### 3.3.2 Accept/Reject Order

- Input:
  Order object forwarded by the management for approval.

- Output:
  Order Status attribute of the order is modified by the agent to success if he accepts the order,
- Process:
  The agent has a waiting time of 15 minutes once the order has been received till which he can accept the order. Beyond which point the order is considered to be rejected by the agent.

## 3.4 User Class – Food Provider

### 3.4.1 Register

- Input: Registration Details
  - Authentication Certificate
  - Location
  - Contact Number
- Output:
  Registration confirmation whether the registration was successful or not.
- Process:
  The food provider delivers all his details to the management during the process of registering which in turn is stored in the database.

### 3.4.2 Enter Menu Details

- Input: Menu Details
  - Food Items
  - Cost of Food items
  - Time for preparation
- Output:
  The details are stored in the database.
- Process:
  The food provider enters his menu, cost and time required for preparation once he is done with registration.

### 3.4.3 Modify Menu

- Input:
  New Menu Details to be overwritten.
- Output:
  Modified menu is stored in the database.
- Process:
  The food provider modifies his menu when some of the items are not available, cost revision or changes in time of preparation.

## 3.5 Data Dictionary

| Data Member | Meaning |
|---|---|
| Order::orderTrain | An instance of the type train to which the order should be delivered to |
| Order::StationOfDelivery | An instance of the type station to which the order must be delivered |
| Order::FoodItems | An array containing objects of the FoodItem type. |
| Order::TimeOfDelivery | The time when the order would be delivered to the customer |
| Order::AcceptedState | It is an integer initialized with value '0' and once an agent accepts an order it is set to value '1' |
| Order::getReceipt | The receipt that is generated once the order has been confirmed |
| Customer::Name | A string storing the name of the customer who places the order |
| Customer::ContactNumber | A string storing the contact number of the customer who places the order |
| FoodItem::Name | A string storing the name of the food item selected by the customer. |
| FoodItem::Price | An integer storing the cost of the food item ordered by the customer |
| Agent::Name | Name of the agent registered with the management. |
| Agent::agentFoodSupplier | An instance of the type food supplier with whom the agent is associated with which should be a single food supplier who is registered with the restaurant. |
| Agent::agentNumberOfOrdersAccepted | An integer storing the total number of orders accepted by the agent from the management which includes both successful orders as well as unsuccessful orders. |
| Agent::agentNumberOfSuccessfulOrders | An integer storing the total number of orders that were successfully delivered by the agent. |
| Agent::ContactDetails | A string storing contact details of the agent registered with the management which is provided during the time of registration. |
| FoodSupplier::Name | A string storing the name of the food supplier registered with the management |

| | |
|---|---|
| FoodSupplier::AuthCert | An image storing the authentication certificate provided by the food supplier at the time of registration. |
| FoodSupplier::Menu | An array of the type FoodItem indicating all the food items served by the restaurant. |
| Train::trainName | A string storing the name of the faring train |
| Train::trainNumber | A string storing the train number of the train. |
| Train::trainRoute | An instance of the type TrainRoute which stores the route through which the train fares. |
| Restaurant::restaurantStations | An array storing instance of RailwayStations that it could reach |
| Restaurant::restaurantNumberOfStations | An integer storing the number of restaurants the restaurant can serve to. |
| FoodChain::restaurantStations | An array storing instance of RailwayStations that the food chain could reach |
| FoodChain::restaurantNumberOfStations | An integer storing the number of restaurants the restaurant can serve to. |
| RailwayStation::stationName | A string containing the station name. |
| RailwayStation::stationCode | A string storing the code of the station. |
| TrainRoute::routeName | A string storing the name of the route |
| TrainRoute::routeRailwayStations | An array storing instances of stations that belong to the particular train route. |
| TrainRoute::routeSuccessfulDeliveries | The number of successful deliveries made in this route |
| TrainRoute::routeTotalDeliveries | An integer storing the total number of attempted deliveries in that route |

# 4. External Interface Requirement

## 4.1 User Interface

User Interface is needed for every component of the software. User Interface will be made such that it is easy to understand for a person using for the first time. Most of the screens will be provided with a help menu and the GUI will be made good looking as well as interactive. With the use of swing interface, the exception or error message will be displayed by a standard JMessageDialog box feature of Java. The interaction with the user interface will be made through mouse and keyboard only, no additional interacting devices are required, and the rest hardware requirements are specified below.

### 4.2 Hardware Interfaces

- Hardware: Personal Computer
- Internet Connection: Either LAN connection or Wi-Fi connection

### 4.3 Software Interfaces

- Software Required: The programming will be done in Java using NetBeans ide. So, the latest version of JDK (jdk 1.7 or above must be installed). As the software depends heavily on databases, so MySQL is also required to be pre-installed.
- Operation System: Windows XP or later, Linux, Mac OS.

## 5. Other Requirements

The system should have minimum amount of RAM to run this software (> 64 MB RAM). The system should have connectivity to access databases or should have the database in the same machine. There are no further requirements.