FINAL REPORT

For

Asset Management System

Prepared by

Jobin Joseph (U101116FCS053)
Mamidi Lokesh (U101116FCS063)
PVNSSK Chaitanya (U101116FCS085)
PSK Vamsi (U101116FCS088)
Pradeep Yadav (U101116FCS089)

Software Engineering
NIIT University

25th November 2018

CHAPTER 1

(Software Requirement Specification)

1. Introduction

1.1 Purpose

We are aiming to develop a Hostel Asset Management System, to keep track of all asset particulars that are currently being used in hostels.

1.2 Document Conventions

Priority decreases with increase in numerical denomination.

1.3 Intended Audience and Reading Suggestions

This document is intended for board of organization, Administrative board and the Staff or request generator. This document specifies the detailed structure of our product, that is, features included, purpose, scope, required environment to use, security details etc.

1.4 Product Scope

For a University or campus scale of asset management, it is usually found to be uneasy to follow the flow of assets and their management. With proper flow control, we are in a process to guarantee hour deep measure of information of a particular asset.

1.5 References

We took the reference from the IEEE-SRS template (IEEE-SRS 830-1998) provided by our course In-Charge to prepare this document.

2. Overall Description

2.1 Product Perspective
It is a self-contained product.

2.2 Product Functions
This software provides Admin/Audit Manager to keep the count of every asset and where the assets are assigned. If any asset is moved from one room to another, it allows you to change the location and the track of this transaction is recorded in the asset history, also for a particular asset the status information can be retrieved for the Admin/Audit manager's use.

2.3 User Classes and Characteristics
There will be three types of users to use this software they are:

1) Admin – Oversees all operations and maintains exclusive rights.
2) Audit Manager – oversees staff operations and reports to Admin, holds the right to remove a particular asset.
3) Staff – Oversees basic asset operations and reports to Audit manager as well as Admin.

2.4 Operating Environment
Web Application, requires basic support providing peripherals, with internet access to run on JRE supported machine.

2.5 User Documentation
We will be providing a user manual so that the user can easily comprehend working of this product and for FAQs and troubleshooting we will be providing all online as well as offline support required.

2.6 Assumptions and Dependencies
This software can give you the correct results if the staff who is using this software should update the asset details correctly whenever it was changed.

3. External Interface Requirements

3.1 User Interfaces

The user interface will be consisting firstly of, a login page where the users will be logging in with a UID and the features respective to their authority, that is, in ascending order Staff<Audit Manager<Admin Head, will be given access to. On login page there will be a logo which will be devised for our initiative, a link leading to a help page which will be including troubleshooting and FAQ related resources and a forgot password/reset password link.

Inside login page:

a) <u>Admin Head</u>

i) Creating an Asset
Introducing an Asset to the system.

ii) Adding an Asset
Establishing a new batch of predefined Asset.

iii) Adding Multiple Assets
Establishing a new batch of multiple predefined Assets.

iv) Viewing Asset Status
Determining whether the asset is Working, Damaged, Repairing, cannot be repaired.

v) Updating Asset Status
Re-Assigning asset status after reviewing the Asset particulars.

vi) Viewing Asset Location
Determining current location of an Asset.

vii) Viewing Assets
Retrieving all information about an Asset or a set of assets.

viii) Viewing Asset Shifting History by Date
Retrieving asset location logs over a specified period of time.

ix) Viewing Asset Shifting History by AssetID
Retrieving asset location logs through AssetID.

x) Viewing Asset Shifting History by UserID
Retrieving asset location logs through UserID.

xi) Viewing Comprehensive Report by Date

Complied report of all events and activities related to hostel asset in a specified date interval.

xii) Viewing Comprehensive Report by AssetID

Complied report of all events and activities related to hostel asset through order of AssetID.

xiii) Viewing Comprehensive Report by UserID

Complied report of all events and activities related to hostel asset through order of UserID.

b) <u>Auditor</u>

i) Viewing Asset Shifting History by Date

Retrieving asset location logs over a specified period of time.

ii) Viewing Asset Shifting History by AssetID

Retrieving asset location logs through AssetID.

iii) Viewing Asset Shifting History by UserID

Retrieving asset location logs through UserID.

iv) Viewing Assets

Retrieving all information about an Asset or a set of assets.

v) Remove Assets

Discarding the Asset particular(s) which is/are rendered beyond repair.

vi) Viewing Asset Location

Determining current location of an Asset.

vii) Viewing Comprehensive Report by Date

Complied report of all events and activities related to hostel asset in a specified date interval.

viii) Viewing Comprehensive Report by AssetID

Complied report of all events and activities related to hostel asset through order of AssetID.

ix) Viewing Comprehensive Report by UserID

Complied report of all events and activities related to hostel asset through order of UserID.

c) **Staff**

i) **Updating Asset Status**
Re-Assigning asset status after reviewing the Asset particulars.

ii) **Updating Asset Location**
Re-Assigning present Asset location of Asset particulars after shifting or related operations.

iii) **Viewing Asset Location**
Determining current location of an Asset.

3.2 Hardware Interfaces

Minimum Requirements for host system are:

A) 64-bit Operating System
B) 2 GB RAM
C) Network Card
D) 8 GB ROM

3.3 Software Interfaces

The system must consist of:

A) JRE
B) Image Processing
C) MySQL (RDBMS)
D) Web Browser support

4. System Features

4.1 Login

4.1.1 Description and Priority

Staff, Audit Manager, Admin can access features respective to their roles in this software.

4.1.2 Stimulus/Response Sequences

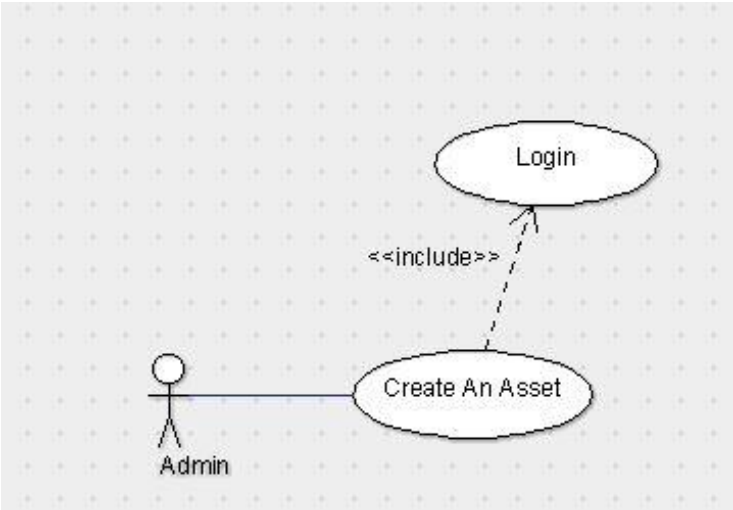
i)Homepage
ii)Entering UID and password
iii)Beginning the session by logging in

4.1.3 Functional Requirements

To act as the bridge to guide different actors to their roles and at the same time protecting the system as a whole against various threats and factors leading to data compromise.

Also providing with user guidelines as well as any and every help required to the actors e.g. help, reset password.

4.2 Create an Asset



4.2.1 Description and Priority

Admin can create and define a new asset if it does not exist yet.

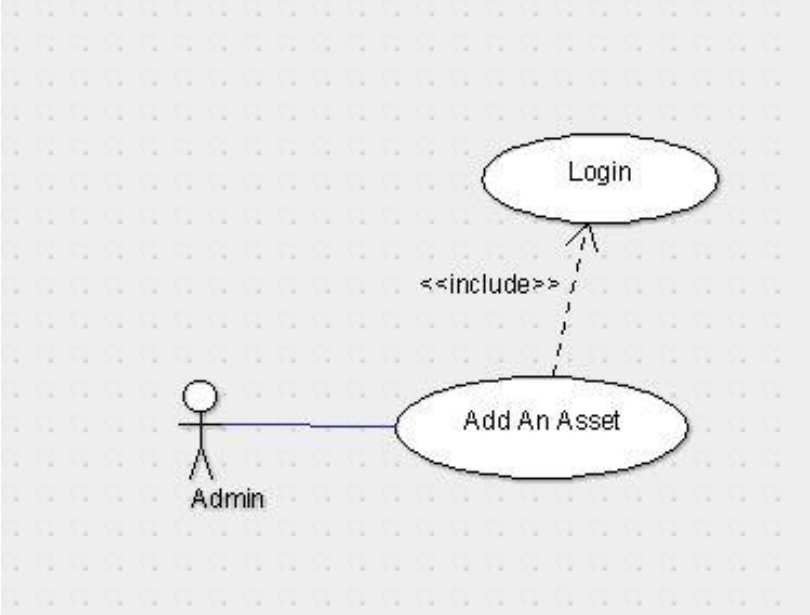
4.2.2 Stimulus/Response Sequence

i) Admin login
ii) Calling Create Asset function
iii) Entering information about the asset entity
iv) Saving changes

4.2.3 Functional Requirements

Checking whether the asset already exists or not, creates asset if it does not exist.

4.3 Add an Asset



4.3.1 Description and Priority

Admin can add pre-defined assets to the current inventory simultaneously assigning a UID to the asset with default location as hostel reception.

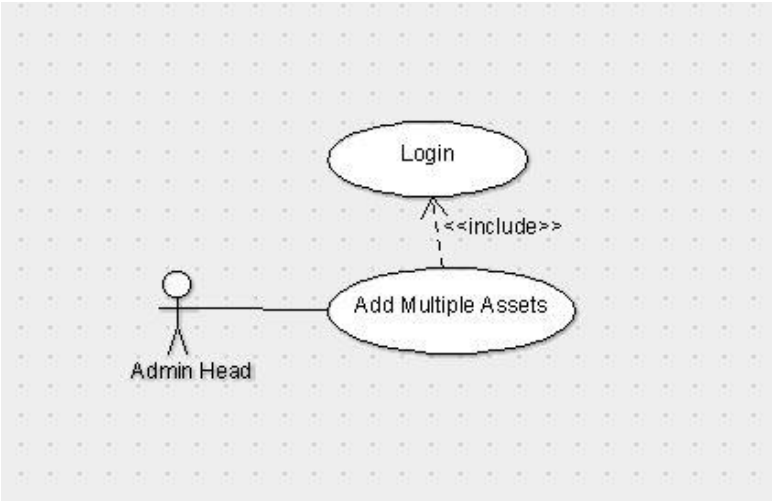
4.3.2 Stimulus/Response Sequence

i)Admin login
ii)Calling Add Asset function
iii)choosing asset type and other necessary details from the drop-down menu
iv)Saving changes.

4.3.3 Functional Requirements

Adding singular particular asset at a time committing to the changes the singular particular asset causes.

4.4 Add Multiple Asset



4.4.1 Description and Priority

Extension of the add Asset feature; adding mass asset entry to the inventory at a time.

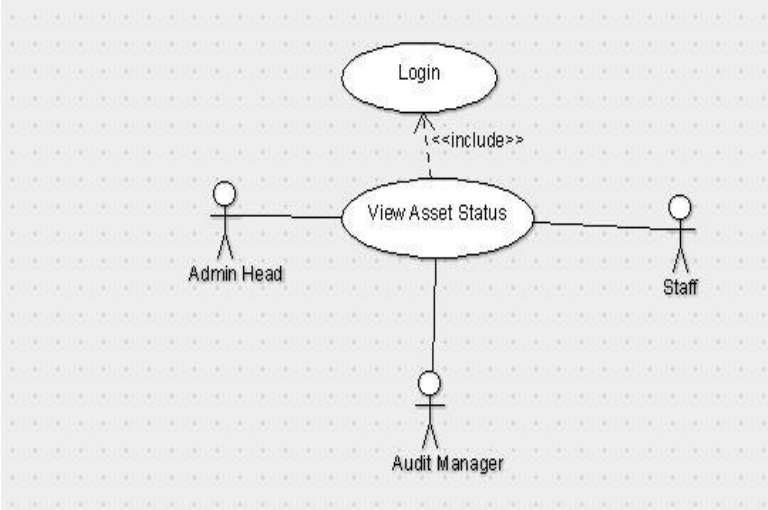
4.4.2 Stimulus/Response Sequence

i) Admin login
ii)Calling Add Multiple Asset function
iii)Selecting asset type from sub-menu
iv)Entering the required number of assets
v) Saving changes

4.4.3 Functional Requirements

Admin is enabled to add multiple asset particulars to the inventory.

4.5 View Asset Status



4.5.1 Description and Priority:

User can see the status of an asset particular i.e. whether it is damaged, working, repairing or cannot be repaired.

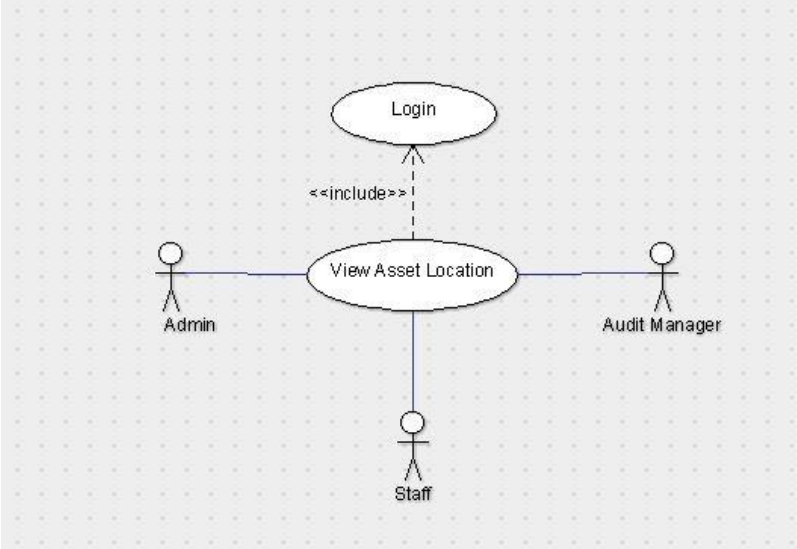
4.5.2 Stimulus/Response Sequence

i)User login
ii)Providing UID to search for an asset particular
iii)Calling View Asset Status function
iv)retrieving asset status.

4.5.3 Functional Requirements

To be able to see the status of an asset particular.

4.6 View Asset Location



4.6.1 Description and Priority

User can view the location of an asset particular.

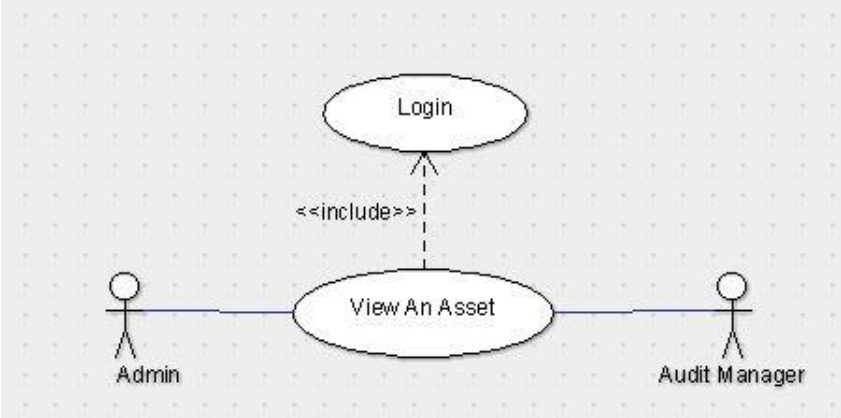
4.6.2 Stimulus/Response Sequence

i)User login
ii)Entering UID of an asset particular.
Iii)Retrieve result.

4.6.3 Functional Requirements

To determine or cross check the location of an asset.

4.7 View Assets



4.7.1 Description and Priority

Admin/Auditor can see the location, status, shifting history and other related information about the asset particular.

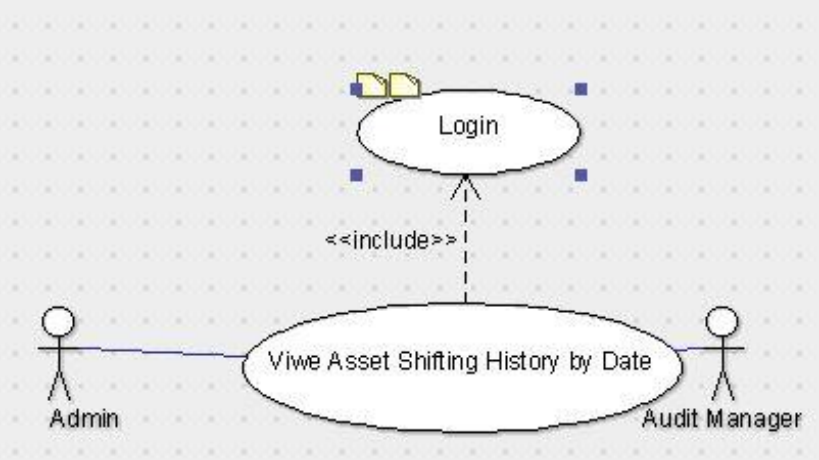
4.7.2 Stimulus/Response Sequence

i) Admin/Auditor login
ii) Providing UID to search for an asset particular
iii) Calling View Asset function
iv) Retrieving asset information

4.7.3 Functional Requirements

To cross check the information related to an asset particular and/or to confirm the existence of an asset particular.

4.8 View Asset Shifting History by Date



4.8.1 Description and Priority

Admin/Auditor are able to see the location history of an asset in the inventory sorted by the date interval.

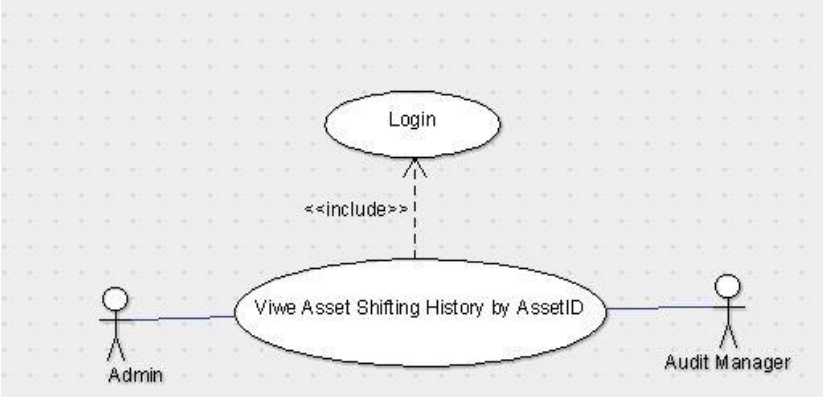
4.8.2 Stimulus/Response Sequence

i) Admin/Auditor login
ii)Entering UID
iii)Specifying date interval
iv)Retrieving result

4.8.3 Functional Requirements

To know the whereabouts of an asset particular in a specified time frame.

4.9 View Asset Shifting History by AssetID



4.9.1 Description and Priority

Admin/Auditor are able to see the location history of an asset in the inventory sorted by the AssetID series.

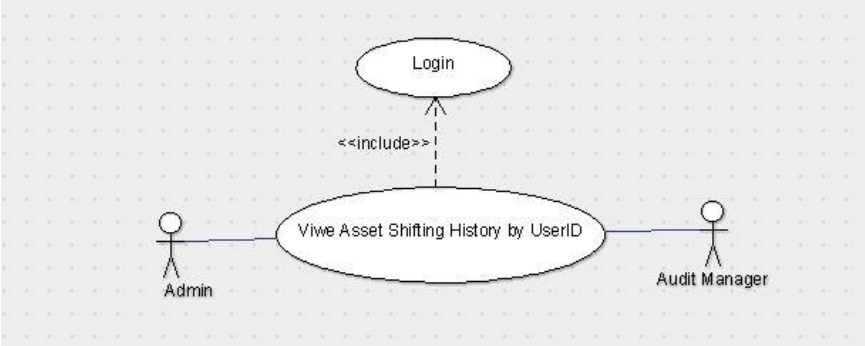
4.9.2 Stimulus/Response Sequence

i) Admin/Auditor login
ii)Entering UID
iii)Specifying AssetID series
iv)Retrieving result

4.9.3 Functional Requirements

To know the whereabouts of an asset particular in a collective of similar assets.

4.10 View Asset Shifting History by UserID



4.10.1 Description and Priority

Admin/Auditor are able to see the location history of an asset in the inventory sorted by the UserID of undertaking User.

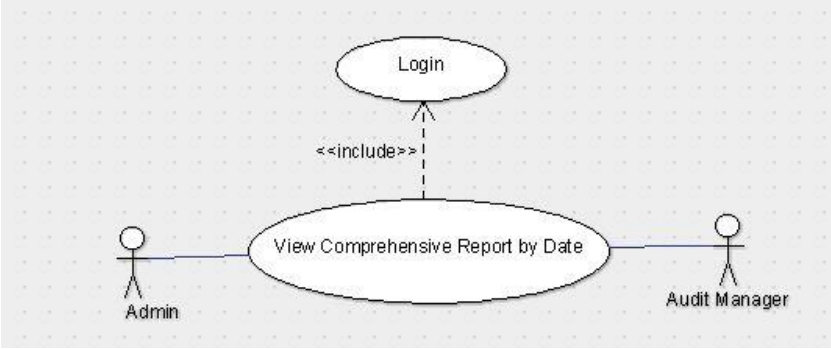
4.10.2 Stimulus/Response Sequence

i) Admin/Auditor login
ii)Entering UID
iii)Specifying UserID
iv)Retrieving result

4.10.3 Functional Requirements

To know the whereabouts of an asset particular in the inventory by operating User through UserID.

4.11 View Comprehensive Report by Date



4.11.1 Description and Priority

Admin/Auditor is able to see the comprehensive report of all operations and activities on the inventory sorted by the date interval.

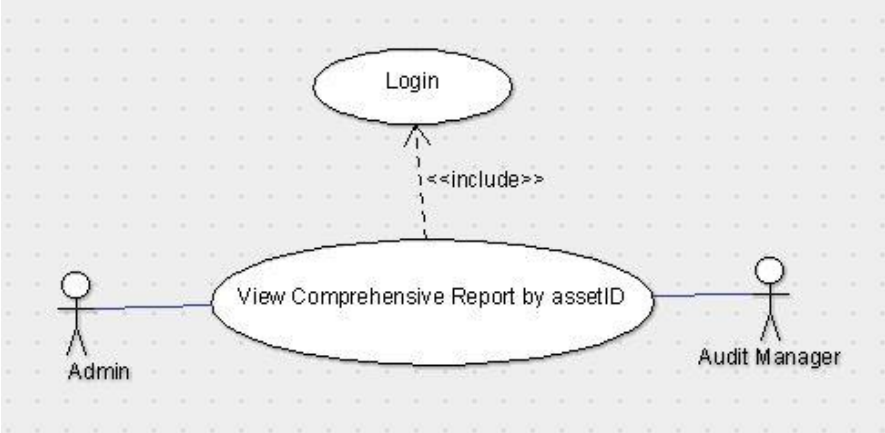
4.11.2 Stimulus/Response Sequence

i) Admin/Auditor login
ii)Entering UID
iii)Specifying date interval
iv)Retrieving comprehensive report

4.11.3 Functional Requirements

To compile all information with proper mapping of events, operations and activities in the specified interval of time.

4.12 View Comprehensive Report by AssetID



4.12.1 Description and Priority

Admin/Auditor are able to see the comprehensive report of all operations and activities on the inventory sorted by the AssetID series.

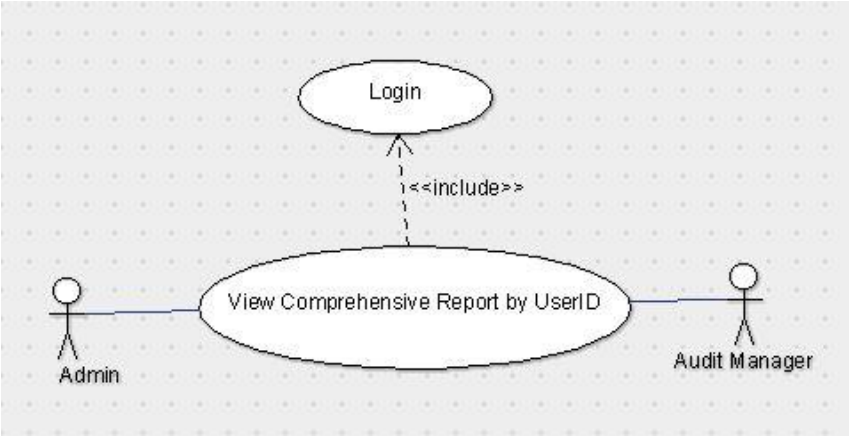
4.12.2 Stimulus/Response Sequence

i) Admin/Auditor login
ii)Entering UID
iii)Specifying AssetID series
iv)Retrieving comprehensive report

4.12.3 Functional Requirements

To compile all information with proper mapping of events, operations and activities on an asset particular in a collective of similar assets.

4.13 View Comprehensive Report by UserID



4.13.1 Description and Priority

Admin/Auditor are able to see the comprehensive report of all operations and activities on the inventory sorted by the UserID of undertaking User.

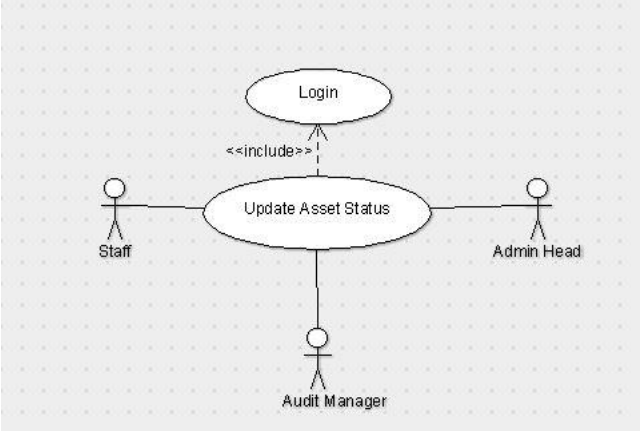
4.13.2 Stimulus/Response Sequence

i)Admin/Auditor login
ii)Entering UID
iii)Specifying UserID
iv)Retrieving comprehensive report

4.13.3 Functional Requirements

To compile all information with proper mapping of events, operations and activities on the inventory by operating User through UserID.

4.14 Update Asset Status



4.14.1 Description and Priority

User can update the status of the asset to whether it is working, damaged or can't be repaired.

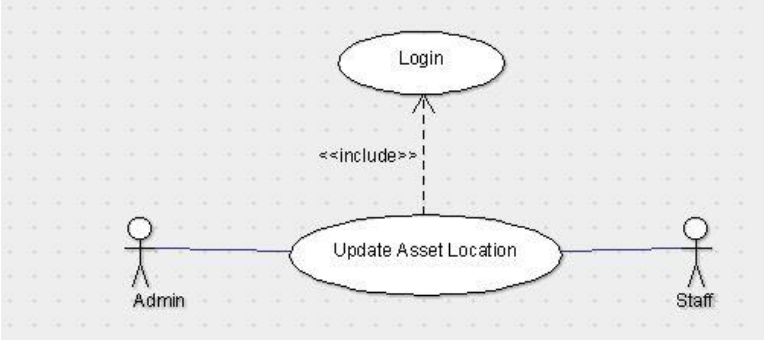
4.14.2 Stimulus/Response Sequences

i)User login
ii)Calling Update Asset Status
iii)Searching an asset particular using UID
iv)Updating status of the asset particular

4.14.3 Functional Requirements

To confirm and update the status of an asset particular after an operation e.g. location update, repair update etc.

4.15 Update Asset Location



4.15.1 Description and Priority

User can update the location of the asset indicating where the asset has been shifted and for what purpose.

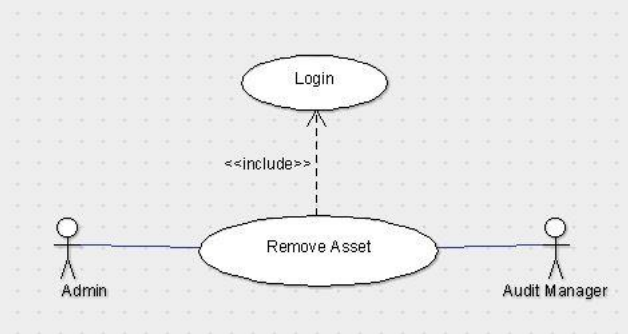
4.15.2 Stimulus/Response Sequences

i)User login
ii)Calling Update Asset Location function
iii)Specifying AssetID
iv)providing necessary information
v)Updating and saving changes

4.15.3 Functional Requirements

To make the system reliable by providing the actors with accurate and efficient information.

4.16 Remove Asset



4.16.1 Description and Priority

Auditor can remove any asset by reviewing the condition of the asset particular if the asset particular is rendered not repairable.

4.16.2 Stimulus/Response Sequences

i)Auditor login
ii)Reviewing asset status
iii)Preparing possible solutions
iv)removing assets that have been rendered unrepairable

4.16.3 Functional Requirements

To review the asset status and for data purification by removing the assets and storing them separately as they are removed.

5. Other Non-functional Requirements

5.1 Performance Requirements

Internet Speed should be responsive and sound.
None of the authoritative actor can login at two different systems during the same session. The host system should be capable enough to support up to 20-30 client nodes in a single session..

5.2 Safety Requirements

The Operational update by the staff user may not tally with the actual facts and information, thus it's the responsibility of the Audit Manager to take care of such wrong transactions.

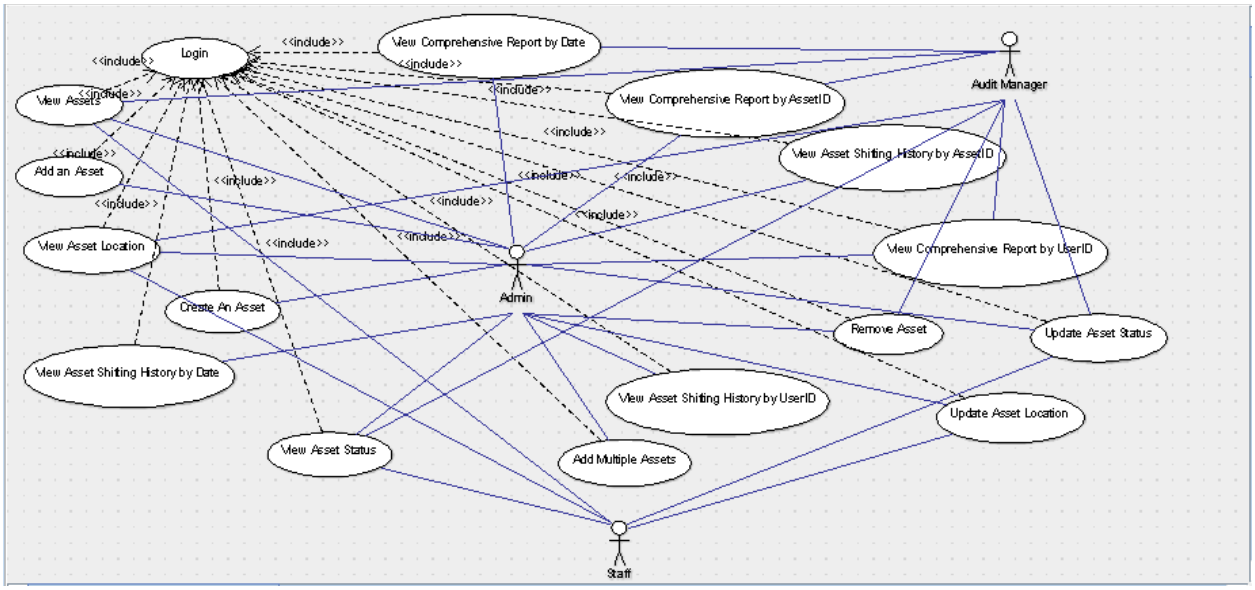
5.3 Security Requirements

User will be provided with a username and password by Admin head to login. We will also be providing separate login credential for separate class of users, that is, an admin will have a separate login ID and staff will have a separate login ID.

5.4 Business Rules

Creating and adding of an asset can only be done by Admin and they can check the asset shifting history, status and location of a particular asset. Audit Manager has the authority to remove an asset, view an asset and can check the shifting history of the assets. Staff can only update and view asset location and status. To protect sensitive information about some of the assets, we are giving different login IDs to different type of users.

Use Case Diagram



CHAPTER 2

(Software Design Specification)

1.1 Introduction

The main purpose of this document is to provide a software development team with overall guidance to the architecture of the software project. This document would help any individual software developer or a software development team to understand the architecture of the project and show the working of a individual software parts as well as the whole software.

1.2 Scope of the development project

For a University or campus scale of asset management, it is usually found to be uneasy to follow the flow of assets and their management.
With proper flow control, we are in a process to guarantee hour deep measure of information of a particular asset.

1.3 Definitions, acronyms, and abbreviations

IEEE: Institute of Electrical and Electronics Engineers
SDS: Software Design Specification

1.4 References

The whole document is built by taking IEEE - SDS template (IEEE 1016)as a reference.

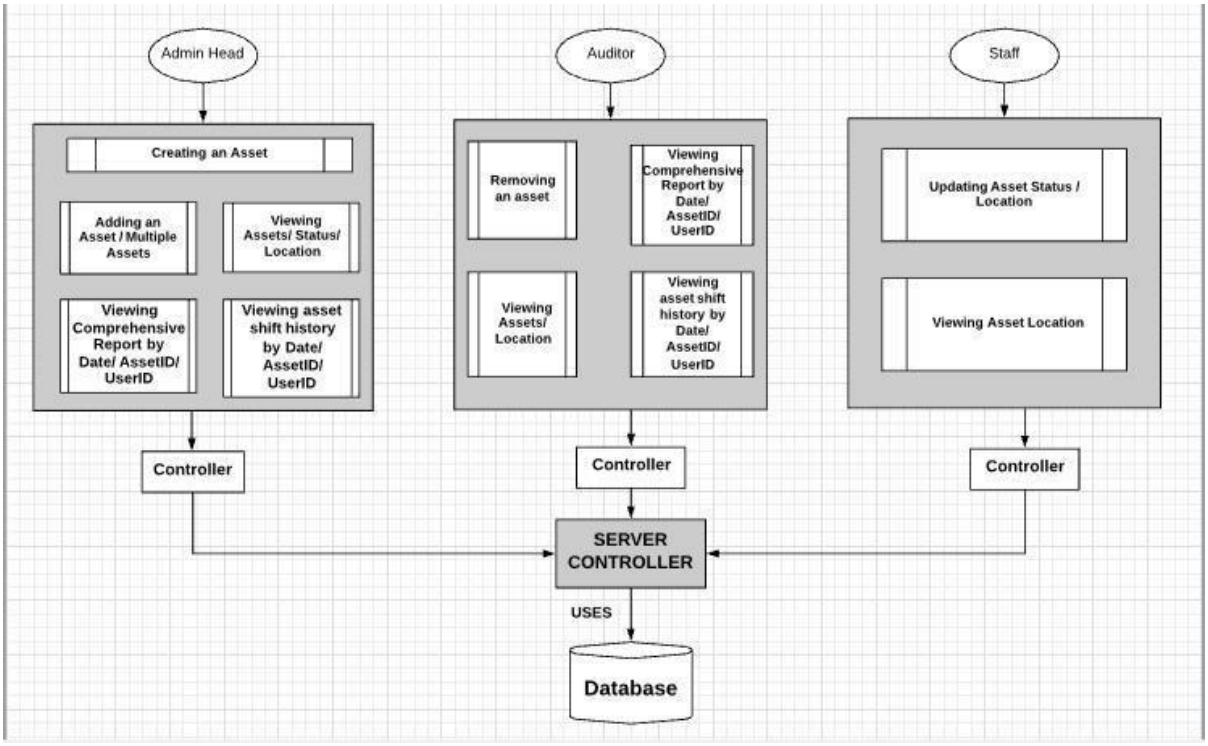
1.5 Overview of document

The SDS is divided into five sections with various sub-sections.
The sections of the Software Design Document are:

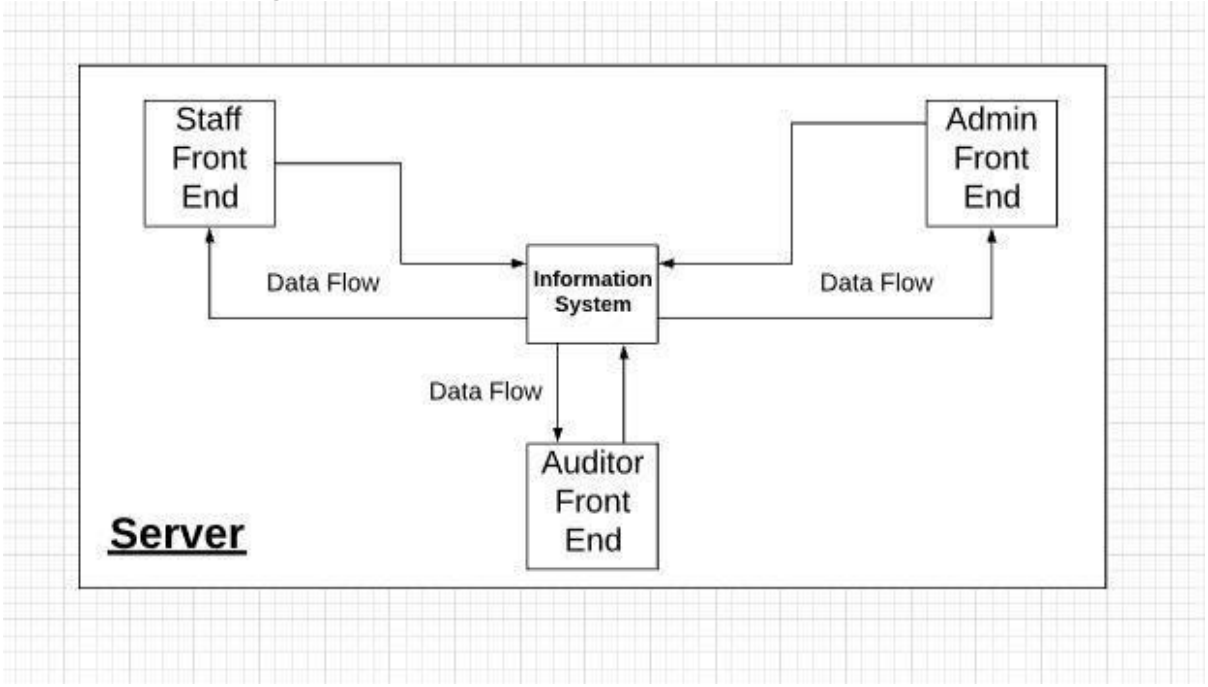
1. Introduction
2. Conceptual Architecture/ Architecture Diagram
3. Logical Architecture
4. Execution Architecture
5. Design Decisions and Trade-offs

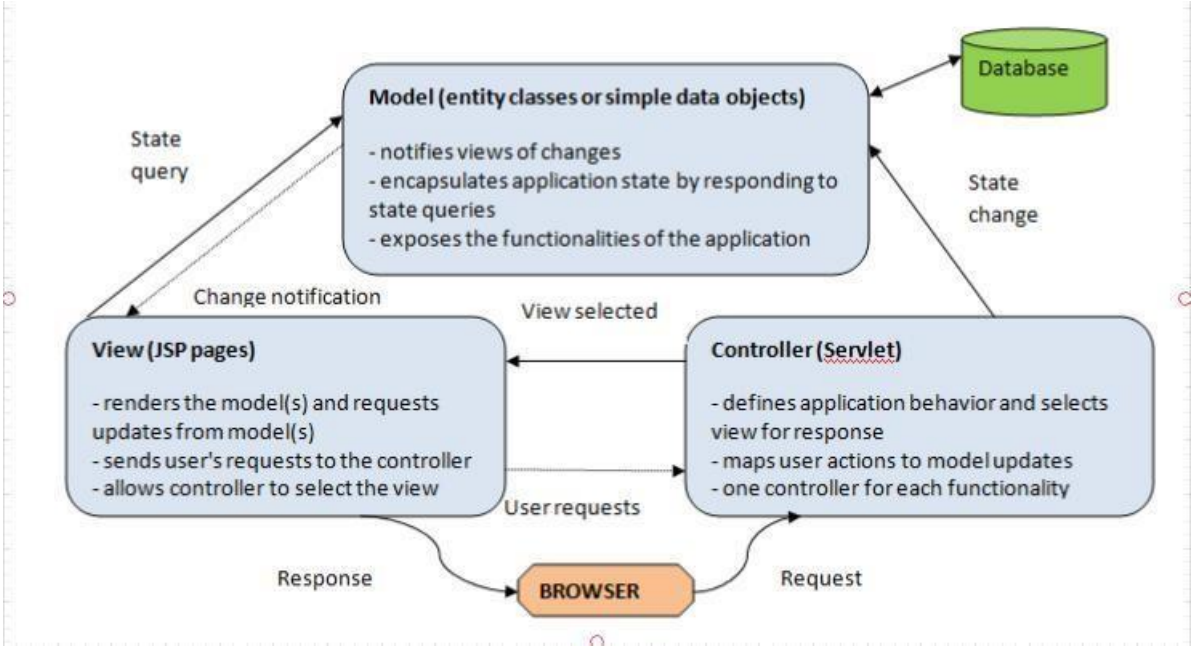
2. Conceptual Architecture/Architecture Diagram

Architecture Diagram 1:



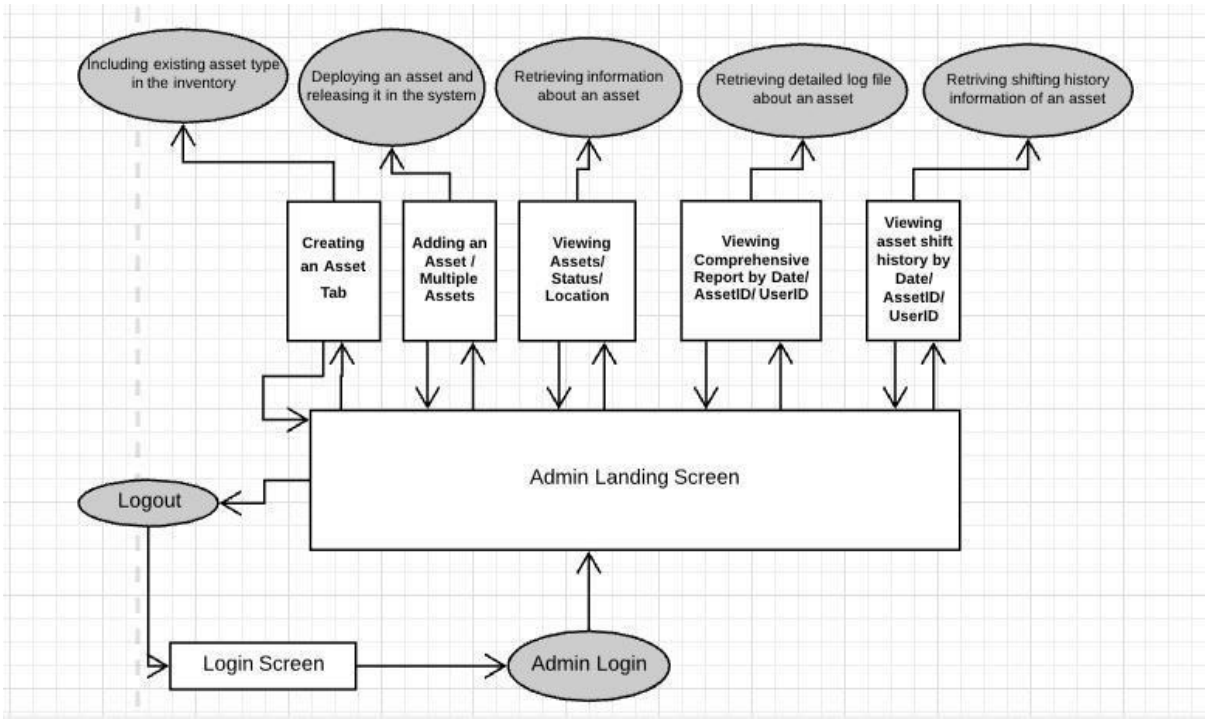
Architecture Diagram 2:



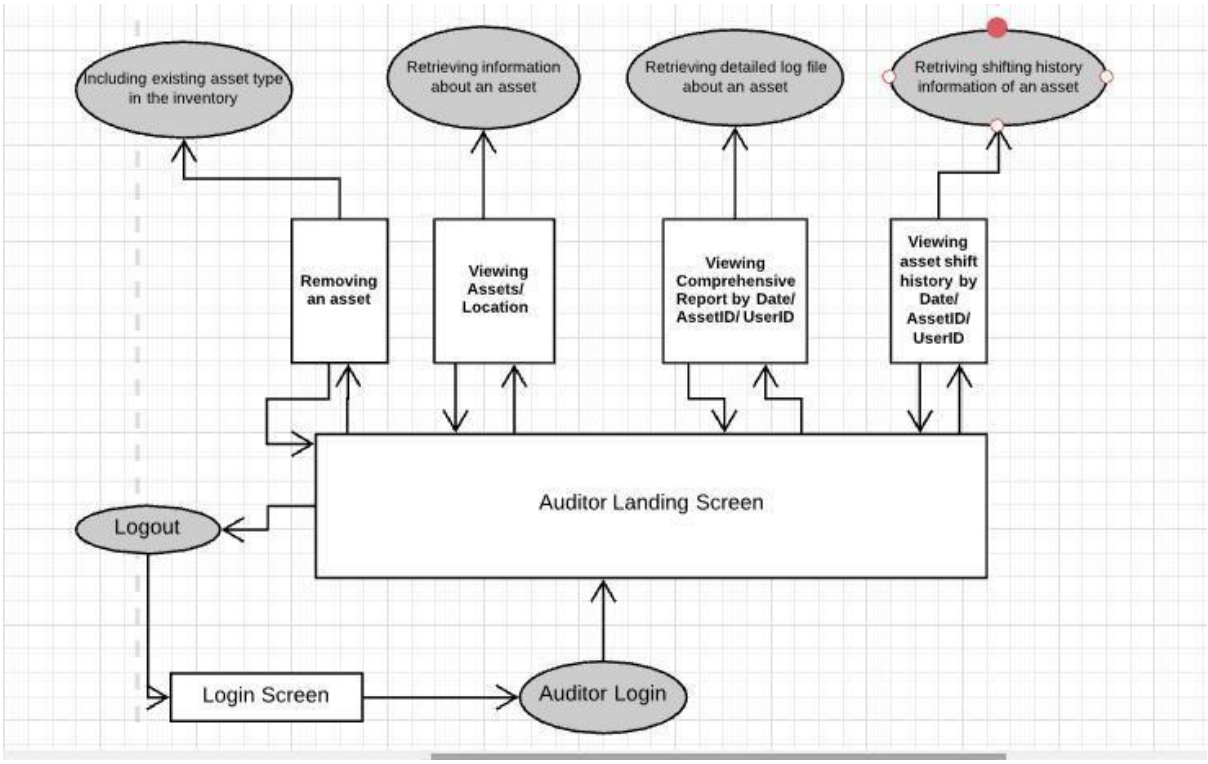


2.2 Structure and relationships

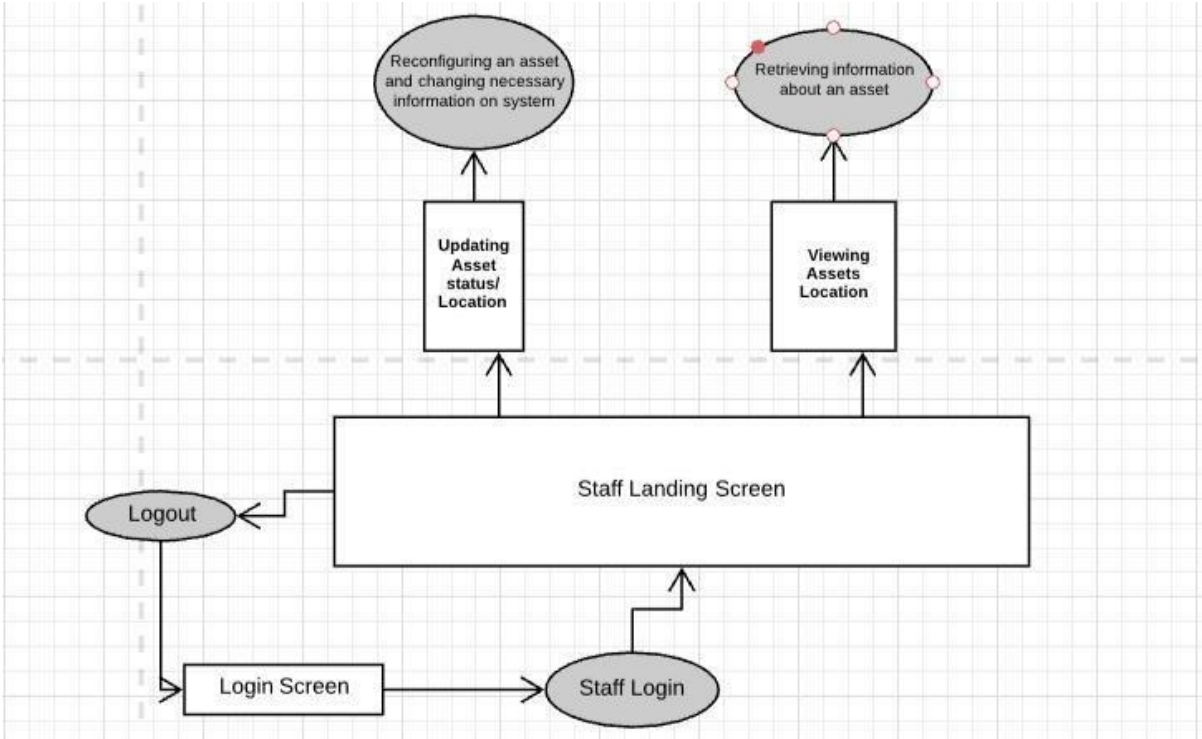
2.2.1 Admin's Side



2.2.2 Auditor's Side



2.2.3 Staff's Side



2.3 User interface issues

This section will address User Interface issues as they apply to the following hypothetical users of the Hostel Asset Management System.

- User X is a 33-year-old male, member of staff, assistant professor at ECE Department at NIIT University. He has been using various computing devices and has a somewhat applicable knowledge of Web-Applications, although he is not "smooth user", he can apply the basic information/instructions given on the user interface to go through the process of viewing asset and updating asset status and/or location and to make the process(es) easier, a hierarchical chain of command and execution oriented user interface is required, aided by messages at stages of required briefing about its actions.

Often this work may seem rigorous and a bit too dynamic hence the user interface is made friendly to the user with careful exposure to the architecture, that is preventing the intimidation and confusion caused by non-supervised release of information.

- User Y is a 36-year-old female employee of the non-teaching staff, who oversees the audit part of the different infrastructural needs at NIIT University.

Her work begins after operations are performed by staff member and/or the admin. Though she isn't proficient at technological advances, she understands the architecture well enough to do her job, this is where we become the middlemen to provide her with ease of access and constructive design of our web-application so as to avoid shortcomings and out of order foundations of the architecture and at the same time being user friendly and less rhetorical.

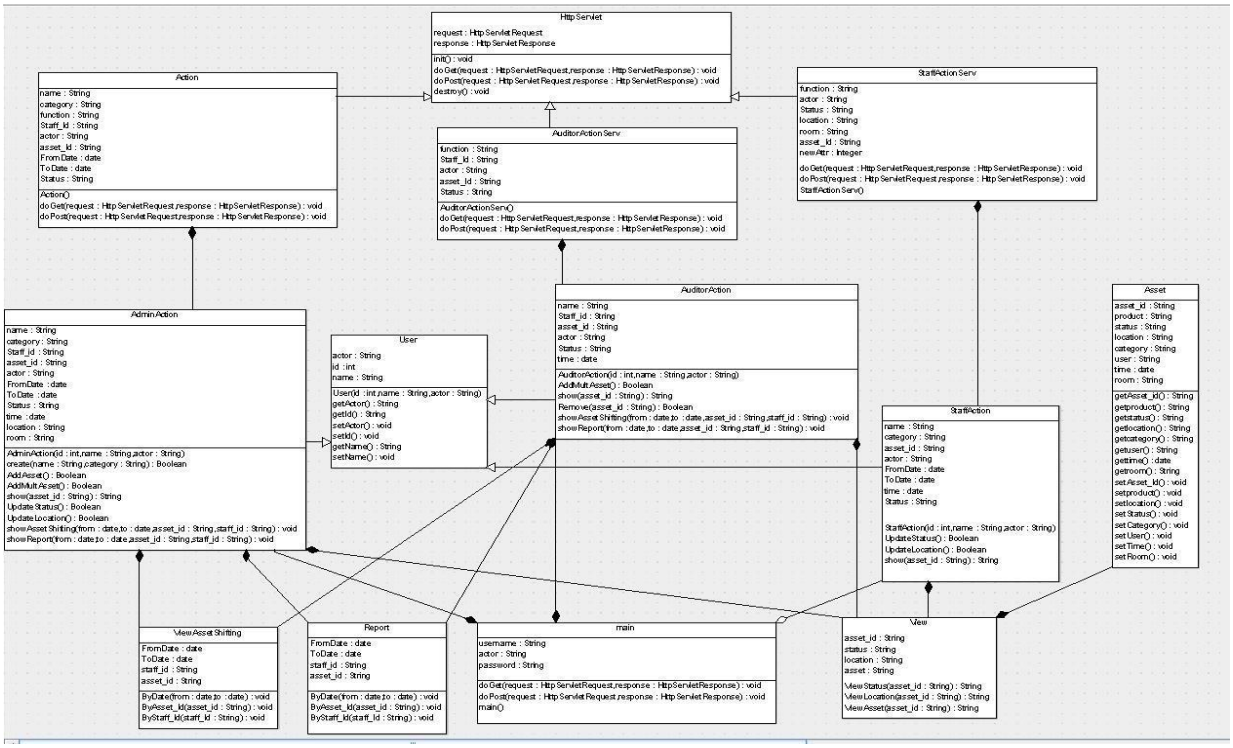
- User Z is a 45-year-old female, head at administrative task force, who oversees all operations under staff and auditor actors and the managing of the Asset management system as whole.

Due to responsibilities and the need of the productive throughput the occurrence of error i.e. unprecedented features or loopholes and system failure is burdening and the reliability of the system is expected to be at par with all acting factors, form-of- actions and feasible forecasts of functionality with administrative clearances and job- functions independent from the descendants (Auditor and Staff).

As the administrator is answerable and responsible to the actions and decisions of the system we tried to make the filing and handling part to be effortless, giving the administrator space to feel free and get a hold of the architecture of Hostel Asset Management System.

3. Logical Architecture Description

3.1.1 Class Diagram:



Class Diagram Description:

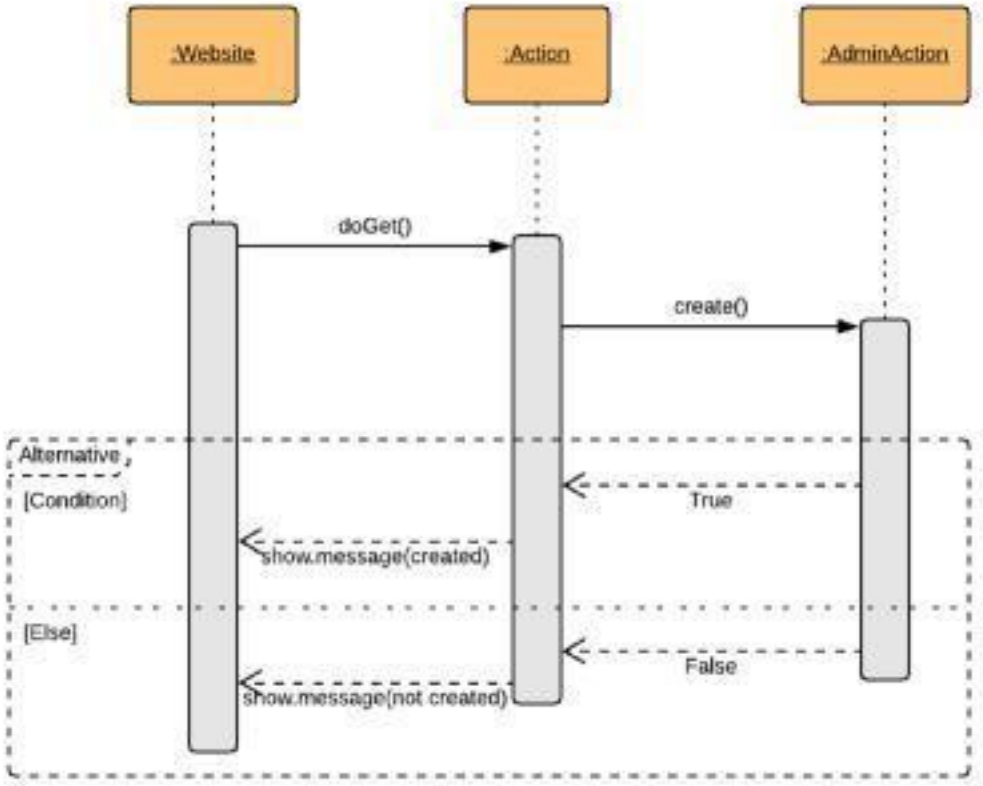
Here, the HttpServlet class is inherited by all the Servlet Classes. User class is inherited by Java classes which have the methods for different actions by different users.

All the methods are discussed class wise in Section 3.2. The class description as well as the different methods along with their Input and output objects and a detailed description in mentioned. The Class diagram here acts as the basic framework as well as the design architecture in the development of the software.

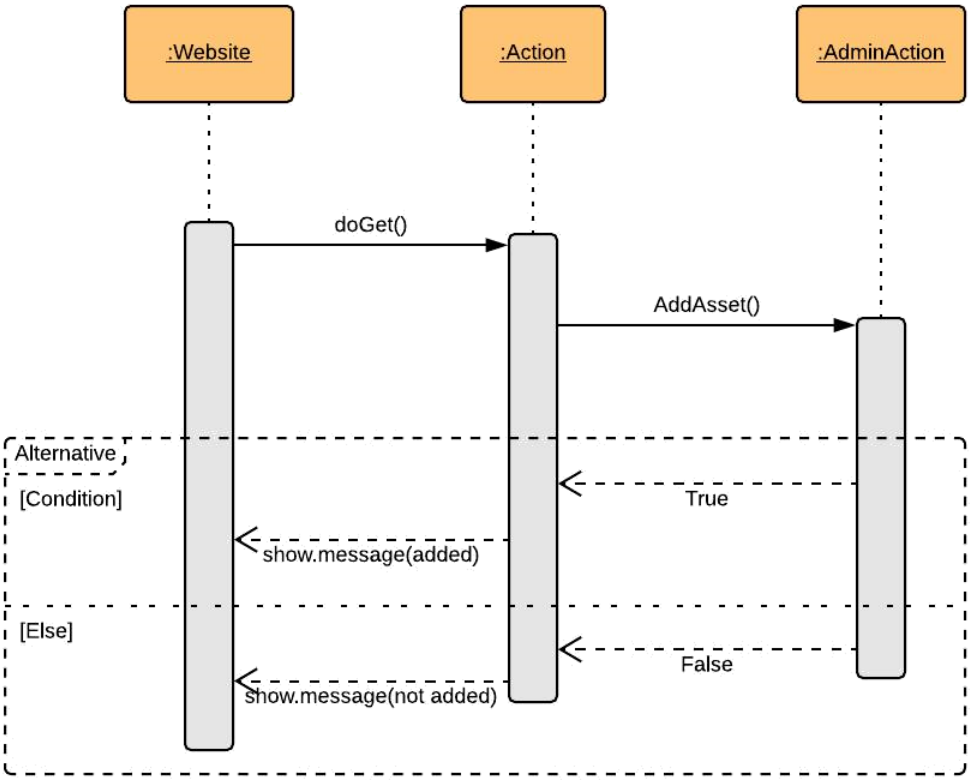
A total of 13 classes are to be observed with different functionality for each class. These classes' methods are called whenever we a specific action is to be performed.

3.1.2 Sequence Diagram:

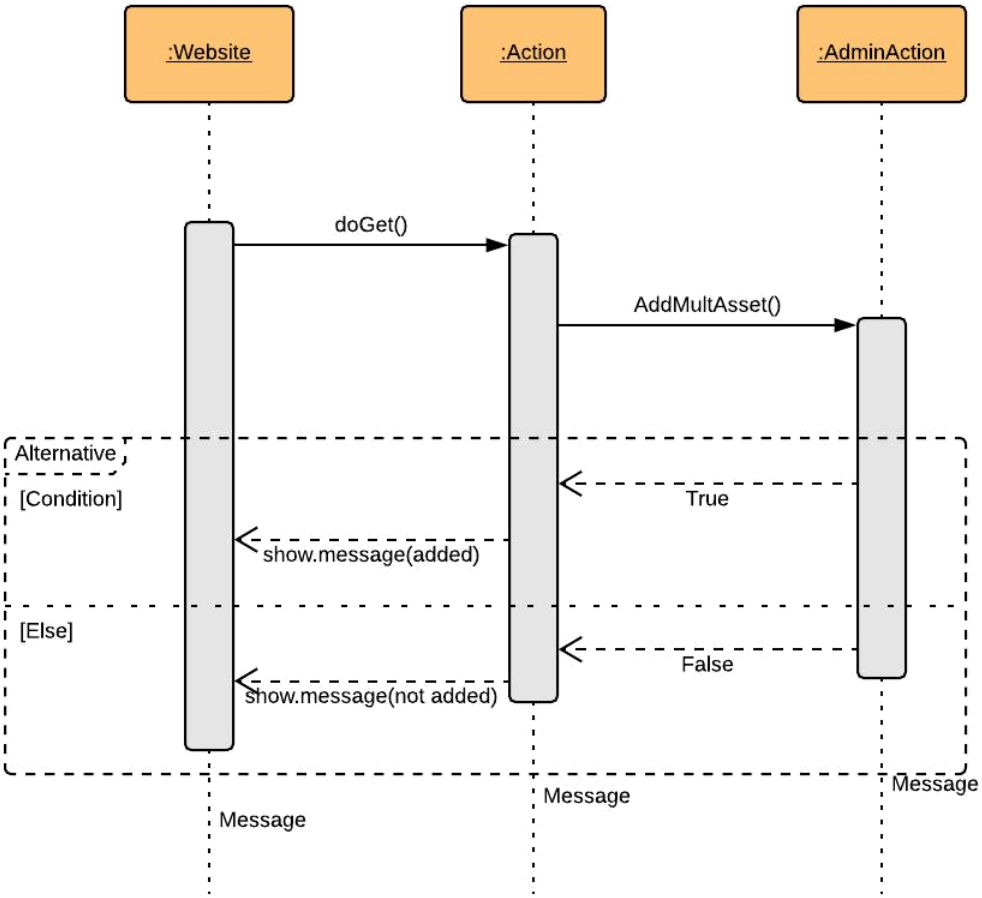
Arrow line signifies there is a send message taken place. Response is being shown by dotted arrows.



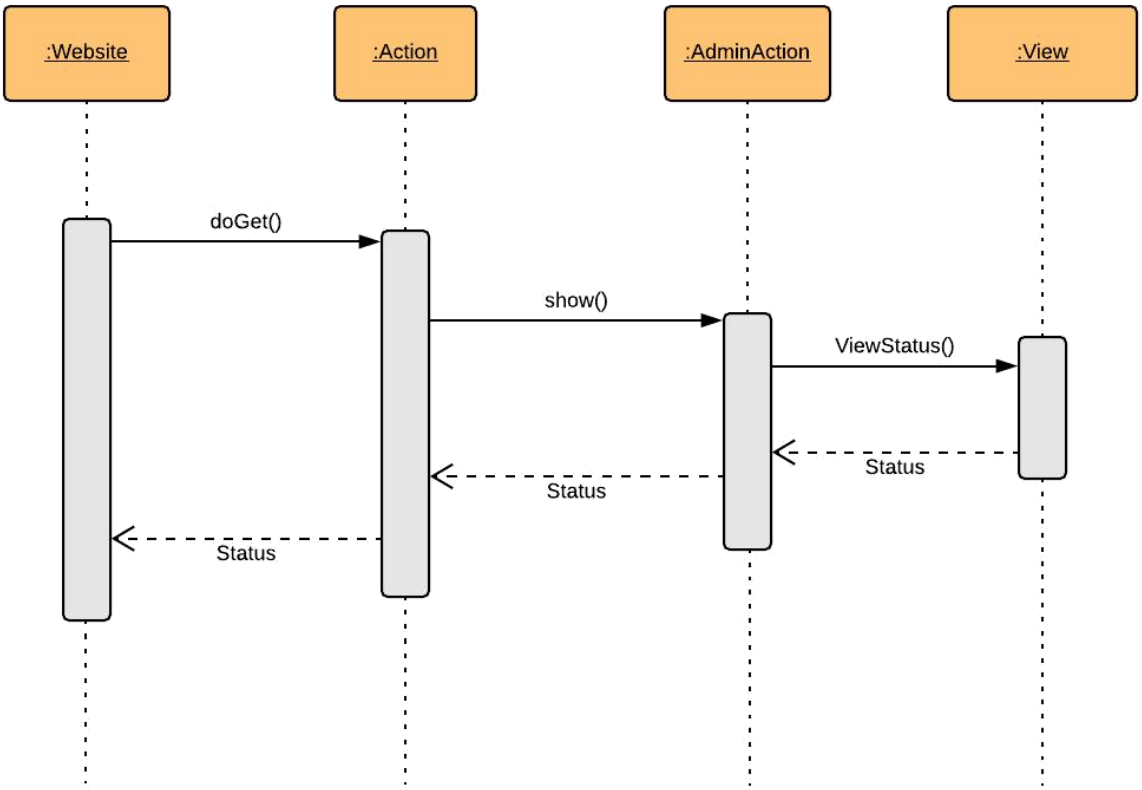
3.1.2.1 Create Asset: The doGet() method calls the create() method which creates an asset and returns a true if created and false in any other case. This same message is shown to the user through a dialog box.



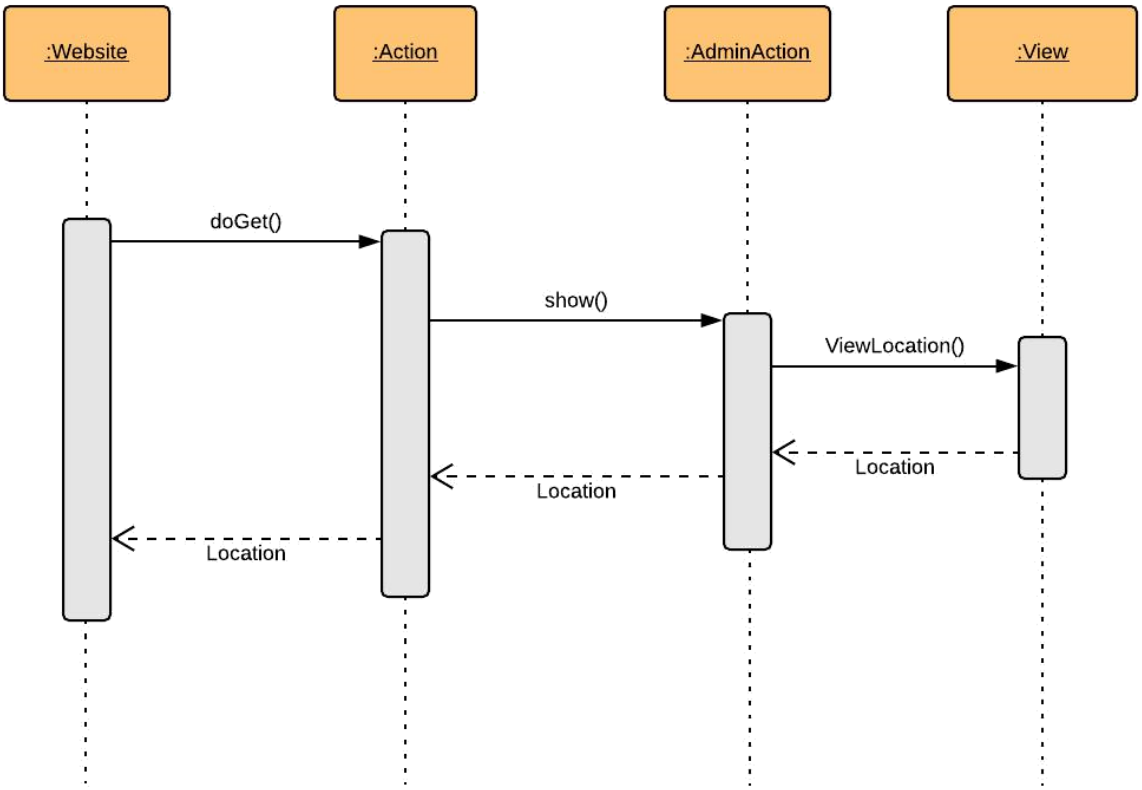
3.1.2.2 Add Asset: The doGet() method calls the AddAsset() method of AdminAction class which adds the details of the asset to the database and returns a message as the previous section.



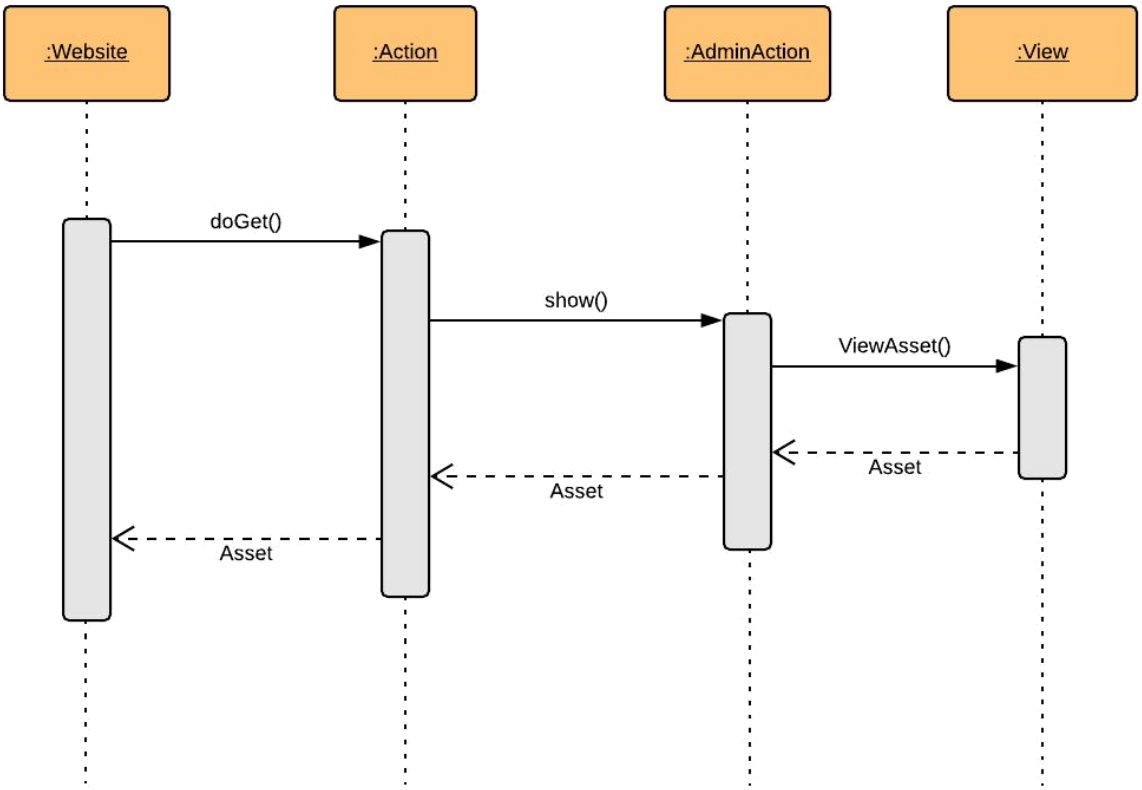
3.1.2.3 Add Multiple Asset: The doGet() method calls the AddMultAsset() method of AdminAction class which adds the details of the assets to the database and returns a message as the previous section.



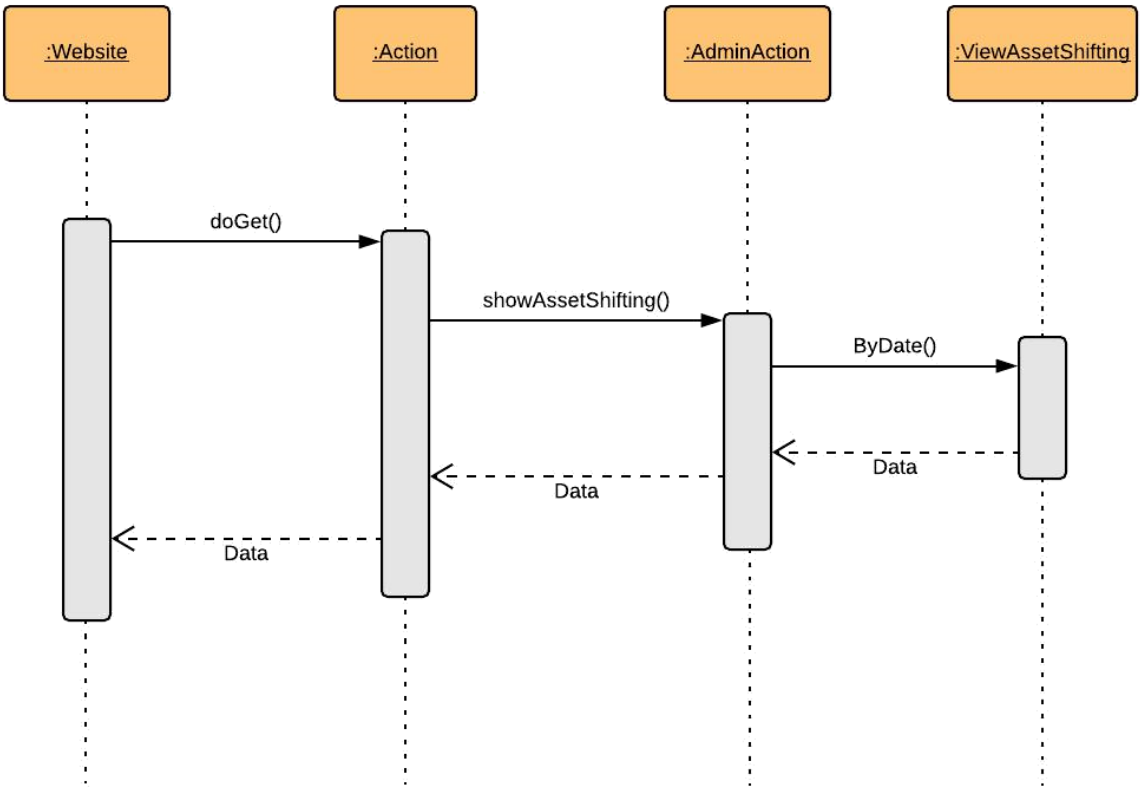
3.1.2.4 View Asset Status: The doGet() method of Action class is called which calls the show() method of AdminAction class which in turn calls the ViewStatus() method of View class. It returns a string with the status of the asset whose assetID is provided.



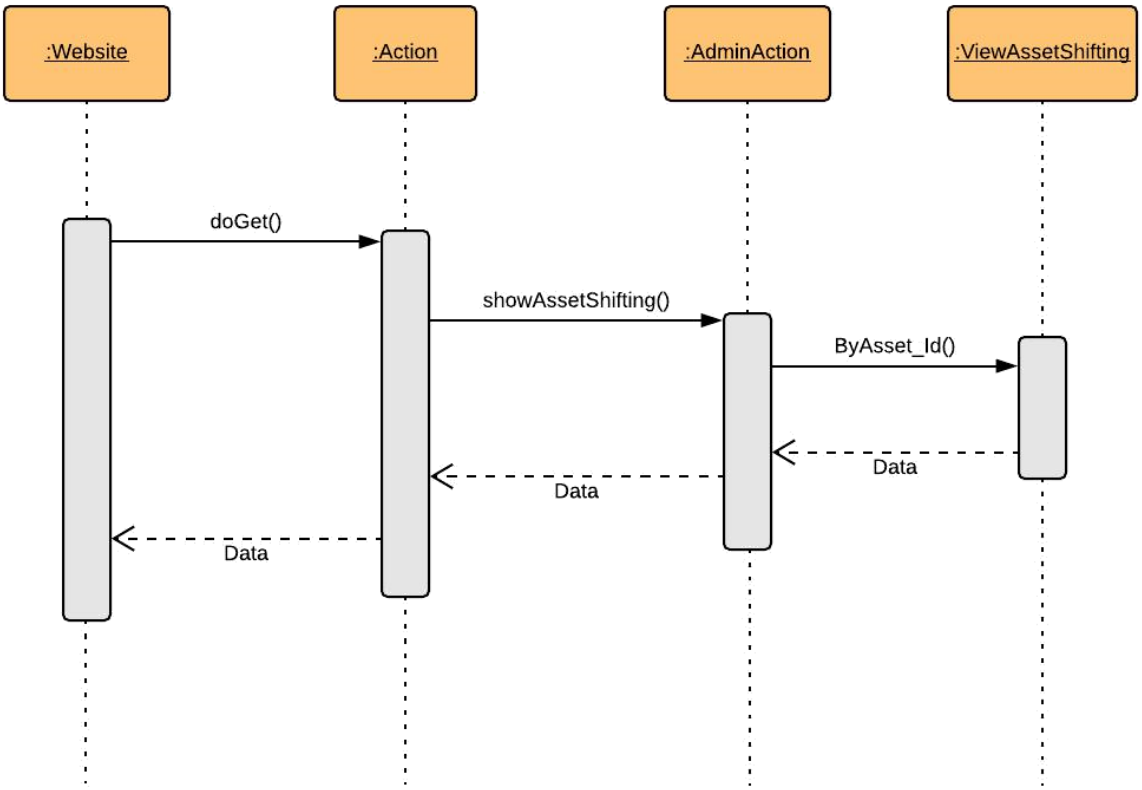
3.1.2.5 View Asset Location: The doGet() method of Action class is called which calls the show() method of AdminAction class which in turn calls the ViewLocation() method of View class. It returns a string with the location of the asset whose assetID is provided.



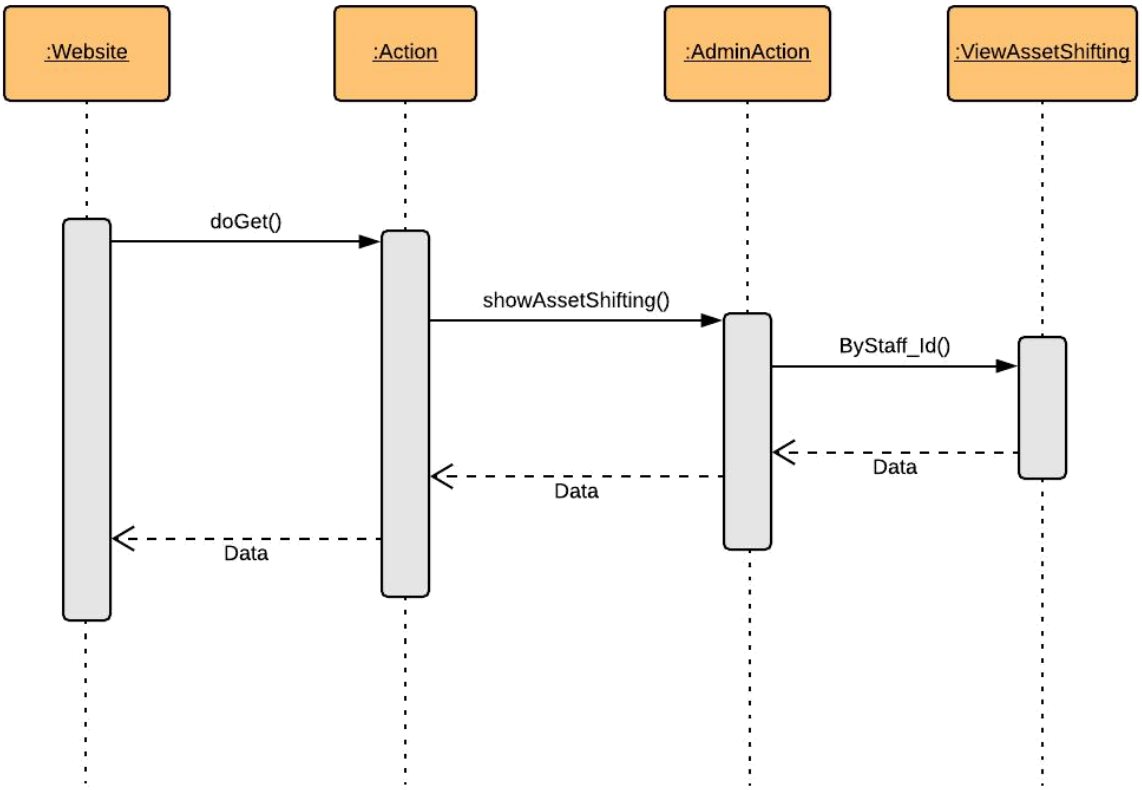
3.1.2.6 View Asset: The doGet() method of Action class is called which calls the show() method of AdminAction class which in turn calls the ViewAsset() method of View class. It returns a string Asset which has all the details of the asset whose assetID is provided.



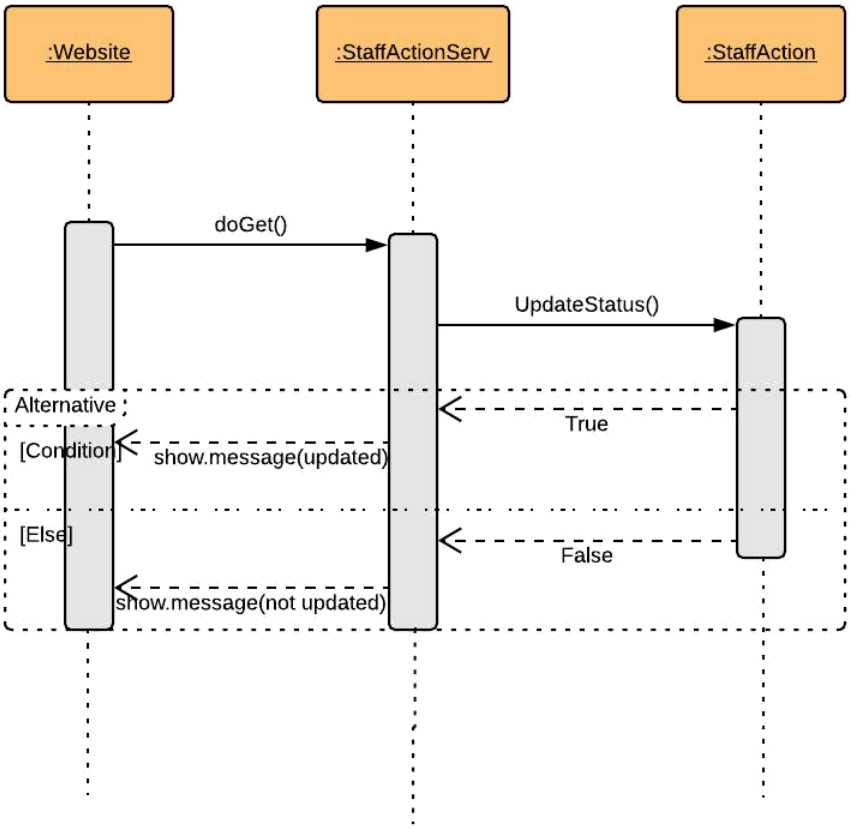
3.1.2.7 View Asset Shifting History by Date: The doGet() method of Action class is called which calls the showAssetShifting() method of AdminAction class which in turn calls the byDate() method of ViewAssetShifting class. It returns a table data which has all the shifting operations from the dates provided.



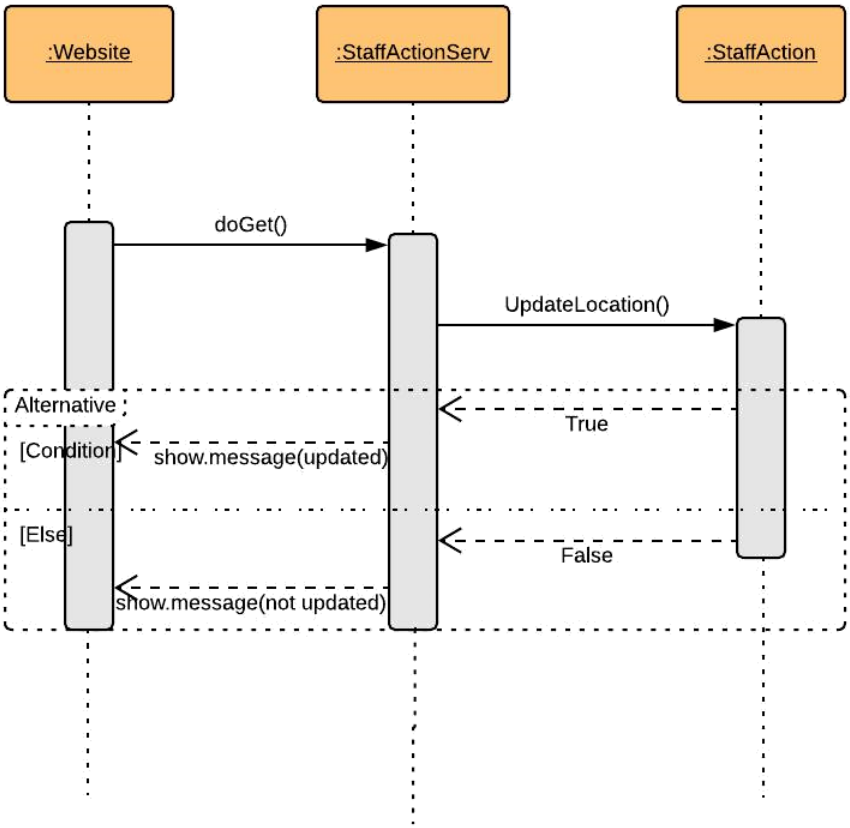
3.1.2.8 View Asset Shifting History by assetID: The doGet() method of Action class is called which calls the showAssetShifting() method of AdminAction class which in turn calls the byassetID() method of ViewAssetShifting class. It returns a table data which has all the shifting operations on a single asset.



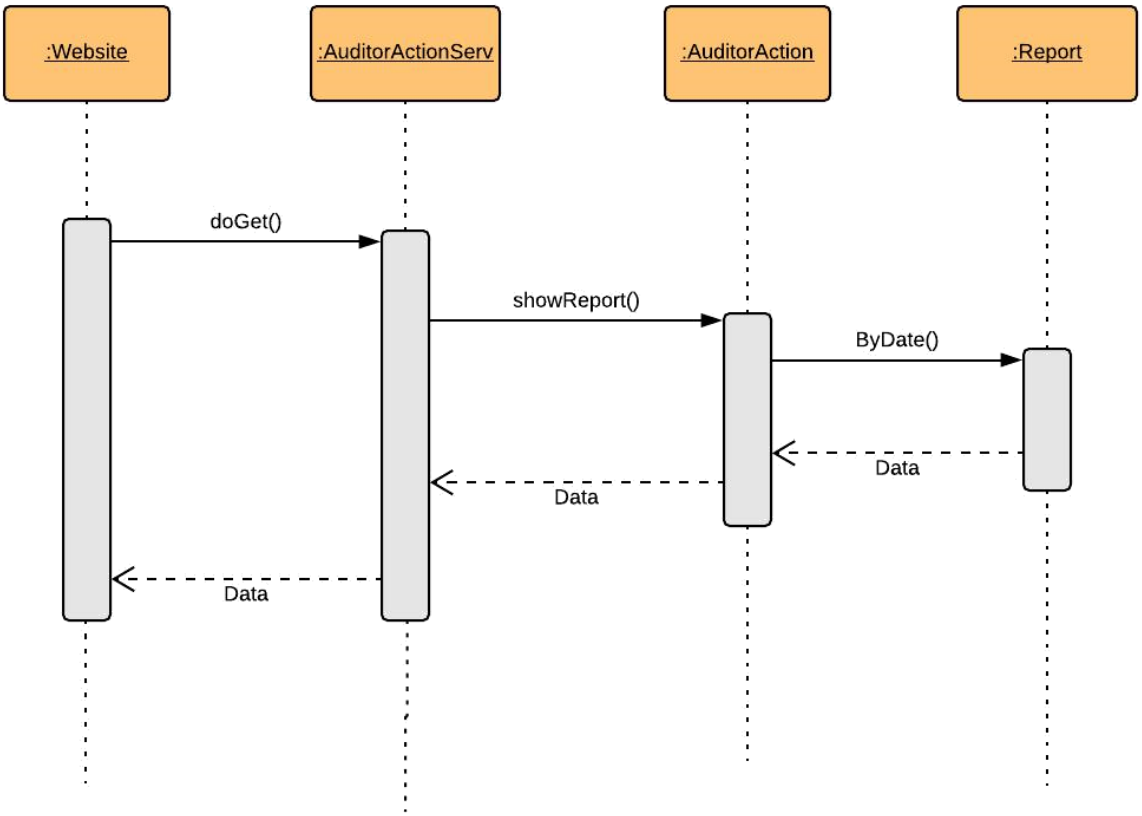
3.1.2.9 View Asset Shifting History by staffID: The doGet() method of Action class is called which calls the showAssetShifting() method of AdminAction class which in turn calls the bystaffID() method of ViewAssetShifting class. It returns a table data which has all the shifting operations by the staff member whose ID is provided.



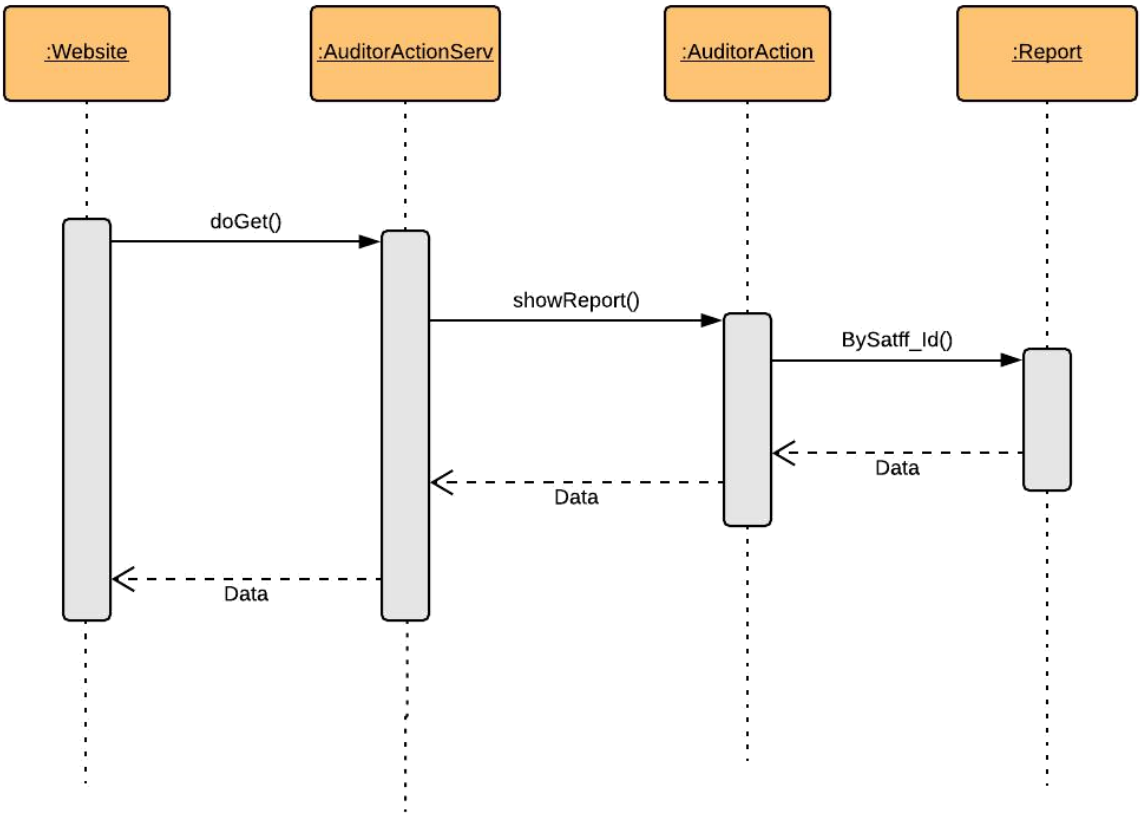
3.1.2.10 Update Asset Status: The doGet() method of StaffActionServ class is called which in turn calls the UpdateStatus() method of StaffAction Class. If the update is successful then, a TRUE is returned else, a FALSE is returned. These values decide the output to the user.



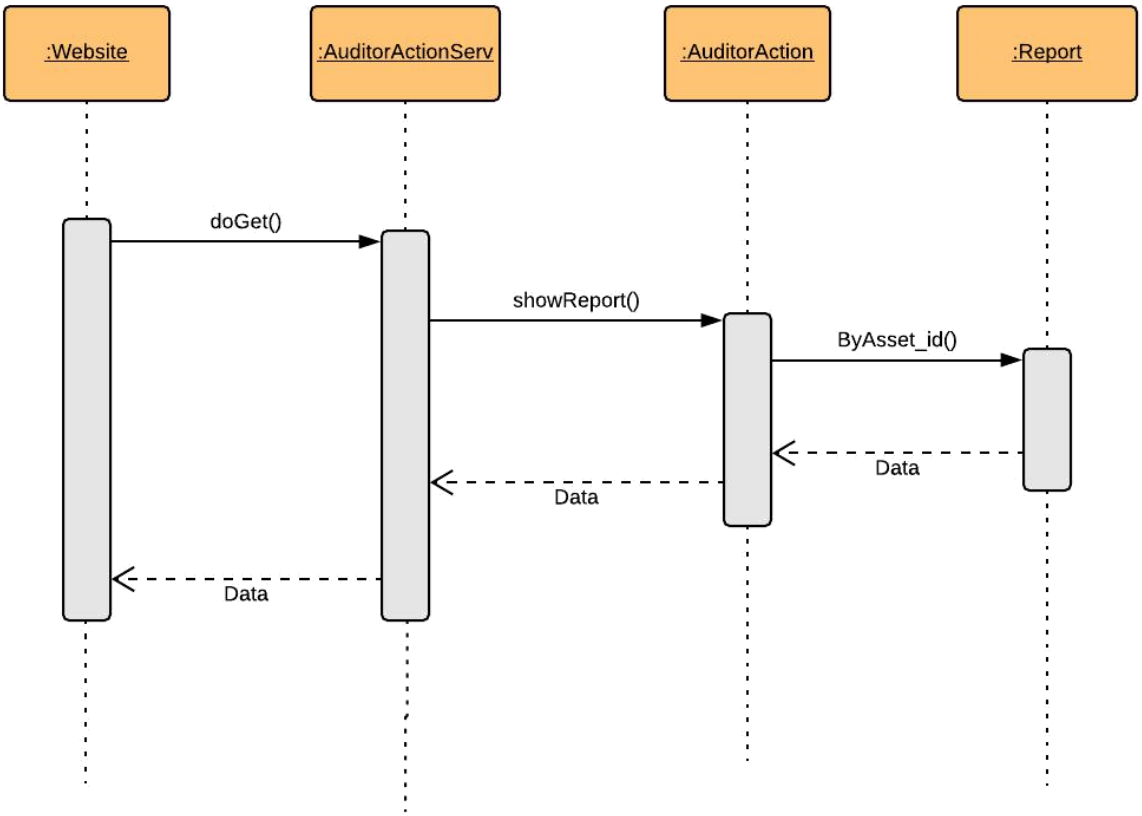
3.1.2.11 Update Asset Location: The doGet() method of StaffActionServ class is called which in turn calls the UpdateLocation() method of StaffAction Class. If the update is successful then, a TRUE is returned else, a FALSE is returned. These values decide the output to the user.



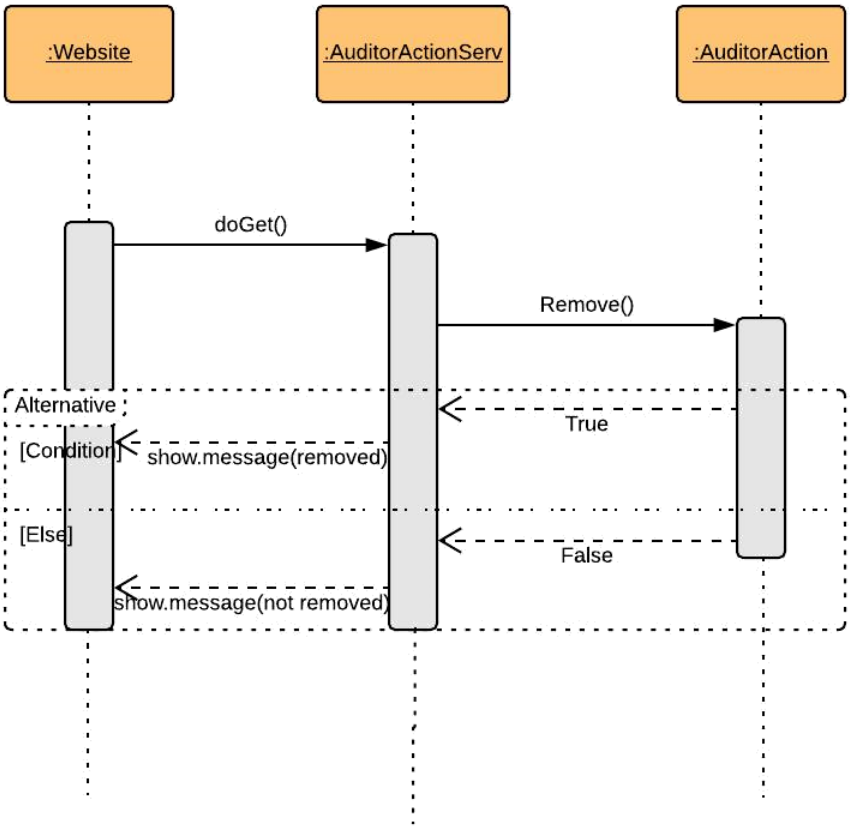
3.1.2.12 Show Report by Date: The doGet() method of AuditorActionServ class is called which calls the showReport() method of AuditorAction Class which in turn calls the byDate() method of the Report class. It returns a table of all the operations between the provided dates.



3.1.2.13 Show Report by StaffID: The doGet() method of AuditorActionServ class is called which calls the showReport() method of AuditorAction Class which in turn calls the byStaffID() method of the Report class. It returns a table of all the operations by the staff member from day of joining to last operation.



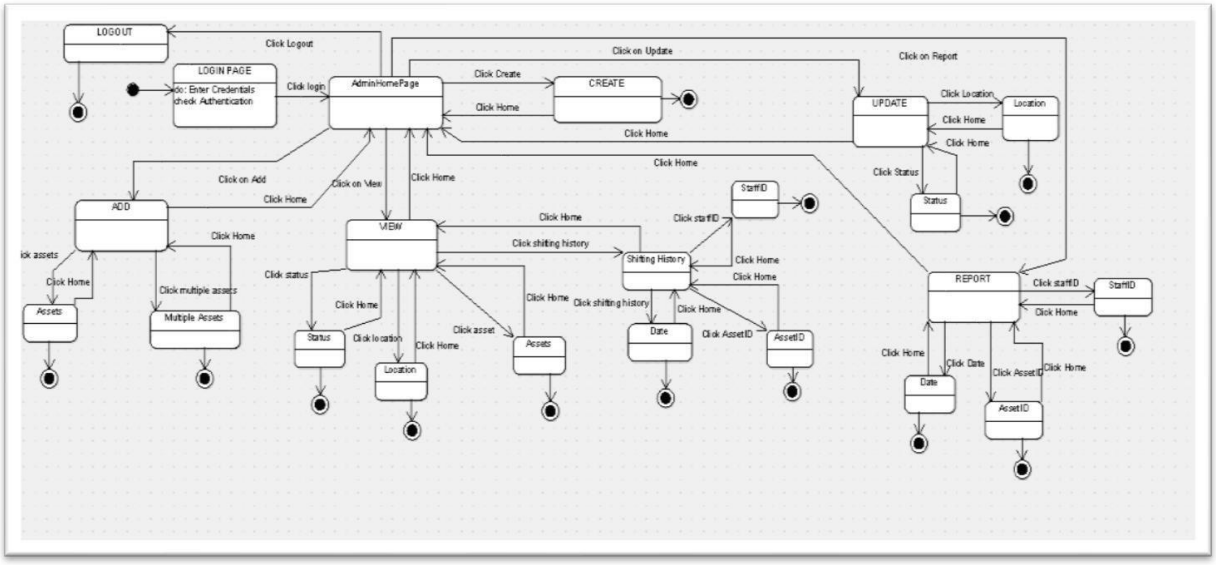
3.1.2.14 Show Report by AssetID: The doGet() method of AuditorActionServ class is called which calls the showReport() method of AuditorAction Class which in turn calls the byAssetID() method of the Report class. It returns a table of all the operations on the asset with the assetID assigned.



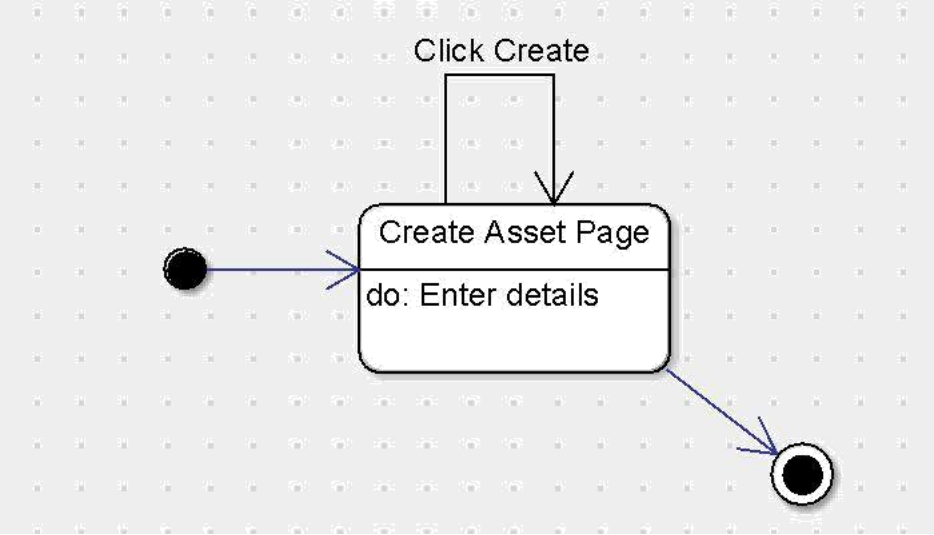
3.1.2.15 Remove Asset: The doGet() method of AuditorActionServ class is called which calls the remove() methods of AudtiorAction class. It removes the asset from the database and returns TRUE if the operation was successful else returns a FALSE.

3.1.3 State Diagram:

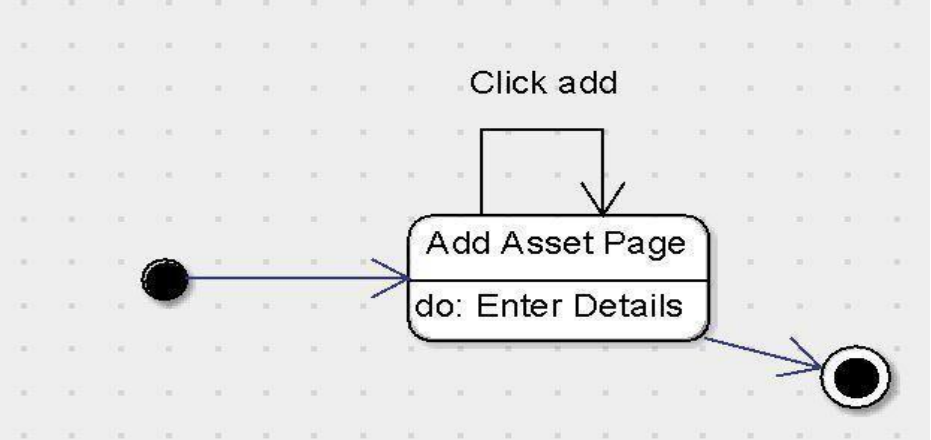
Initial state is being shown by starting with a black dot. Final State is being shown by the black dot surrounded by an empty circle.



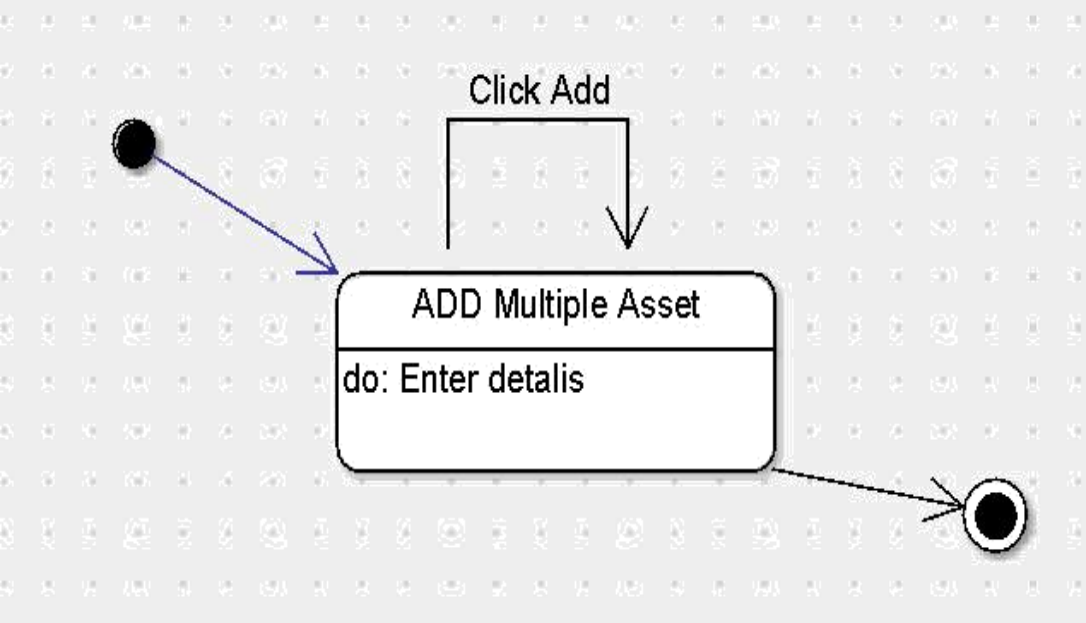
3.1.3.01 Admin Landing Page: This page has all the operations for the user. Here, the user is directed to each and every new page depending the function or use of the application. User can even logout from here. Which is actually signing off the application.



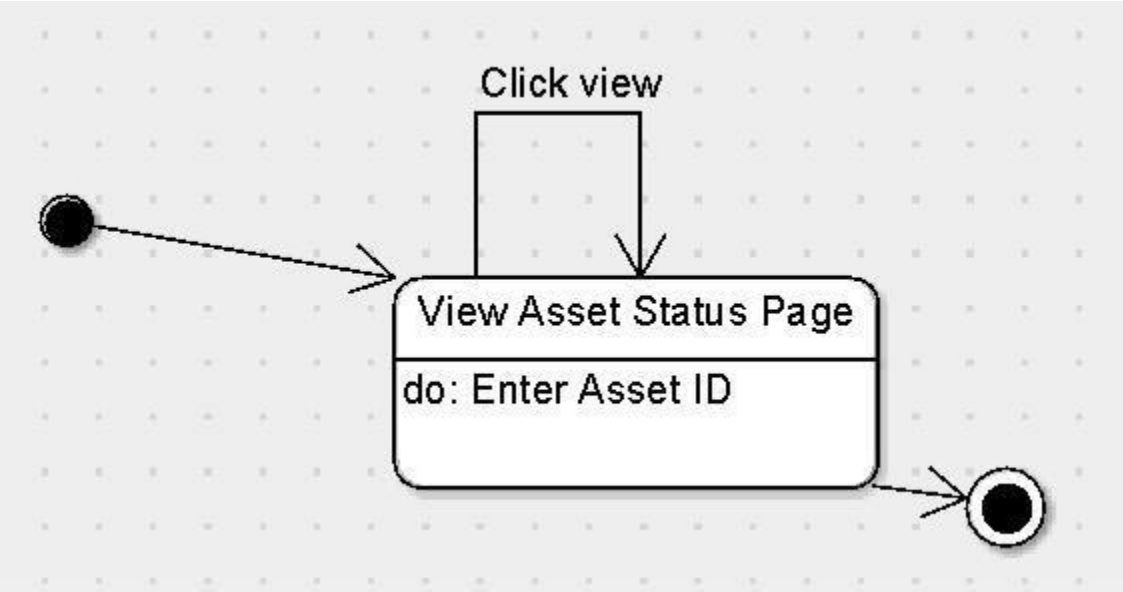
3.1.3.02 Create Asset: It allows user to enter the name as well as the category of the asset which will be added later. Entering data into this page tells to the system that there is a new type of asset coming in. Later, the assets purchased will be added using the add asset page.



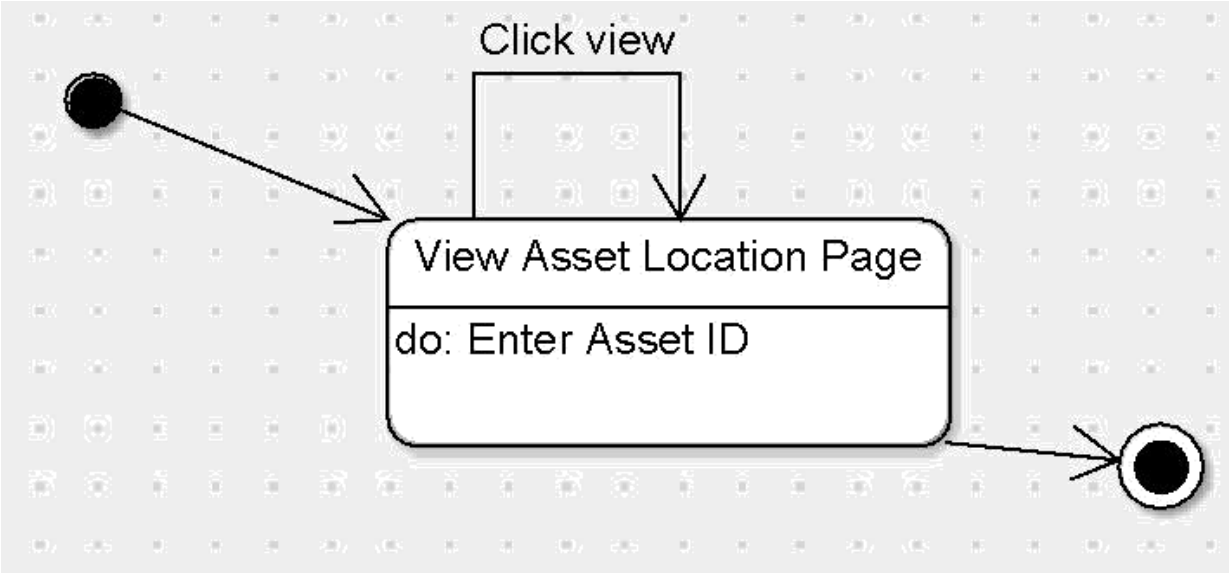
3.1.3.03 Add Asset: It allows user to add an asset which was already created. The user can only add an asset and get the Asset ID of the asset for future reference.



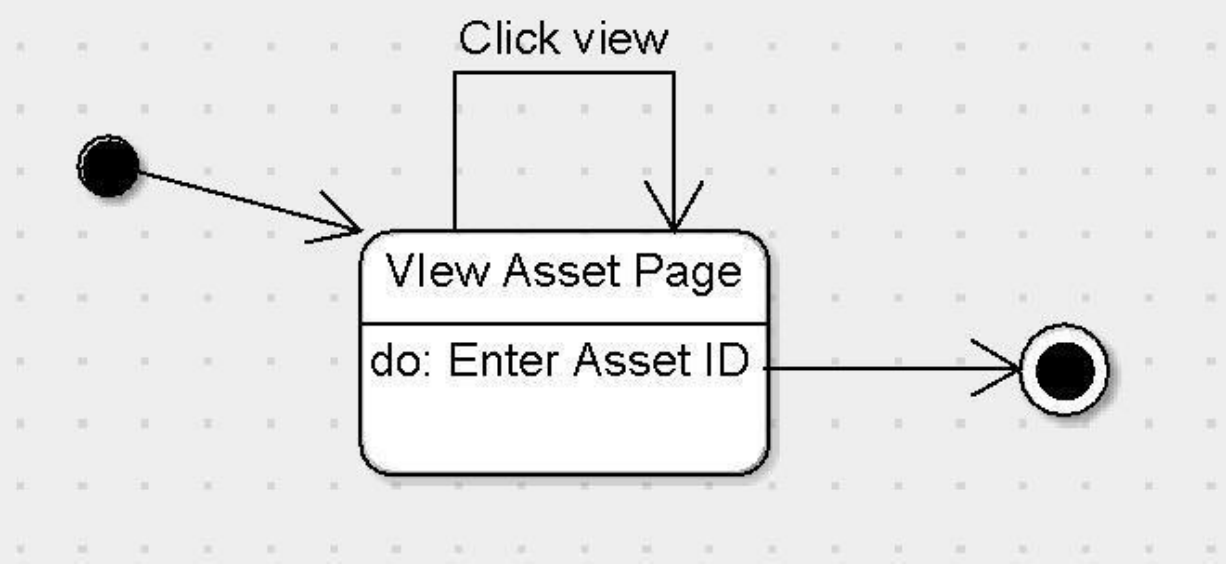
3.1.3.04 Add Multiple Asset: It allows user to more than one asset of same type at a single click. The user provides the Product ID (which is different for each type of asset) and the number of assets to be added. The output would be a set of Asset IDs for each asset for future reference.



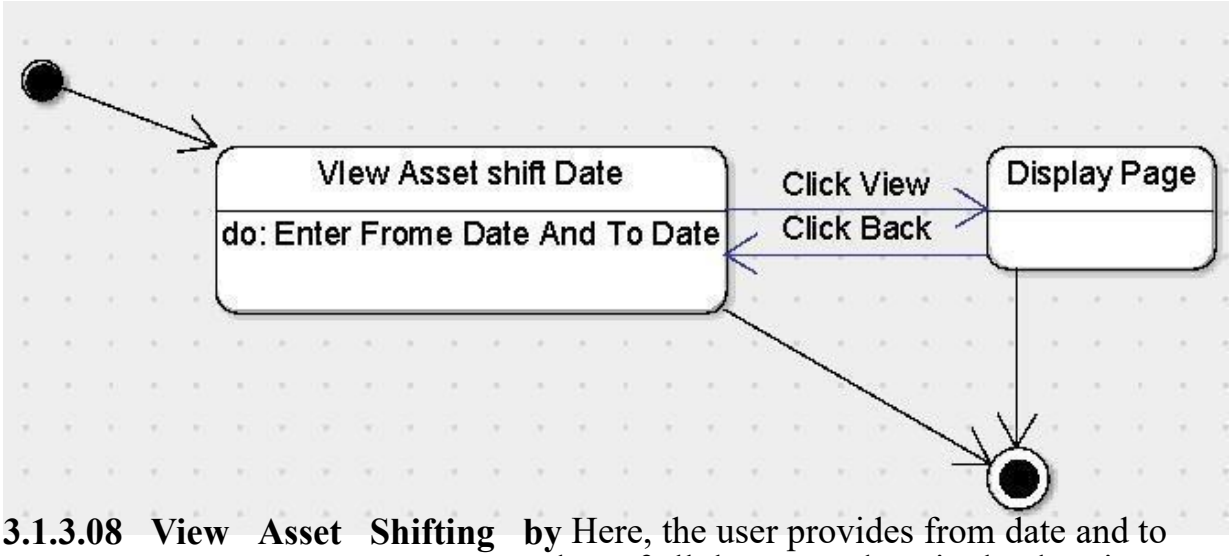
3.1.3.05 View Asset Status: The user gets the status of the asset i.e., broken, can't be repaired, to be repaired, beyond economic rate and many more just by providing the Asset ID of the asset and clicking View button.


Click view
View Asset Location Page
do: Enter Asset ID

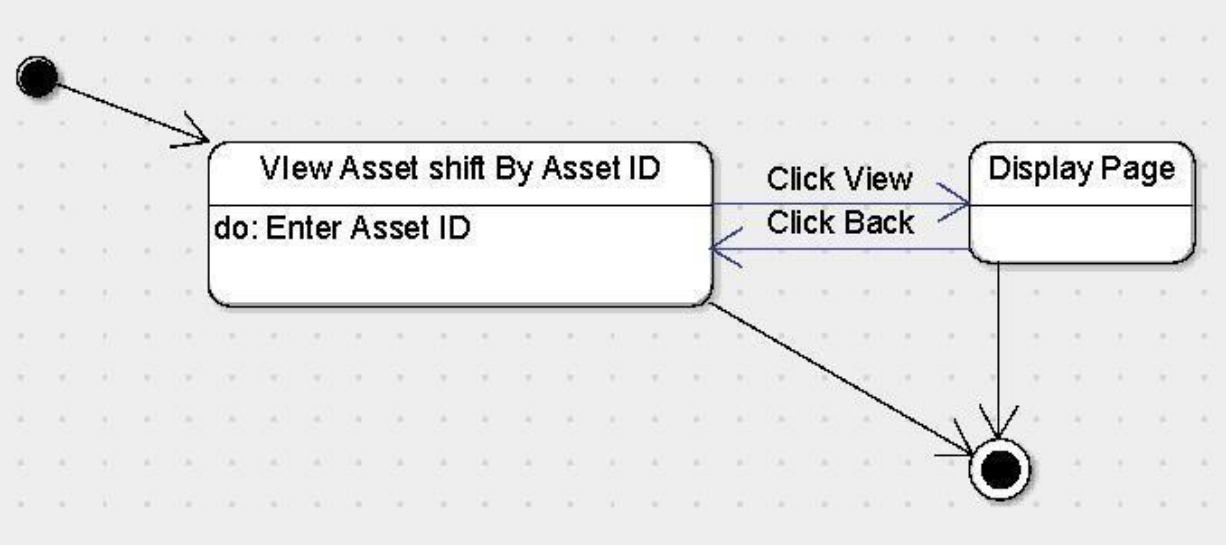
3.1.3.06 View Asset Location: The user gets the location of an asset just by providing the Asset ID of the asset and clicking View button.



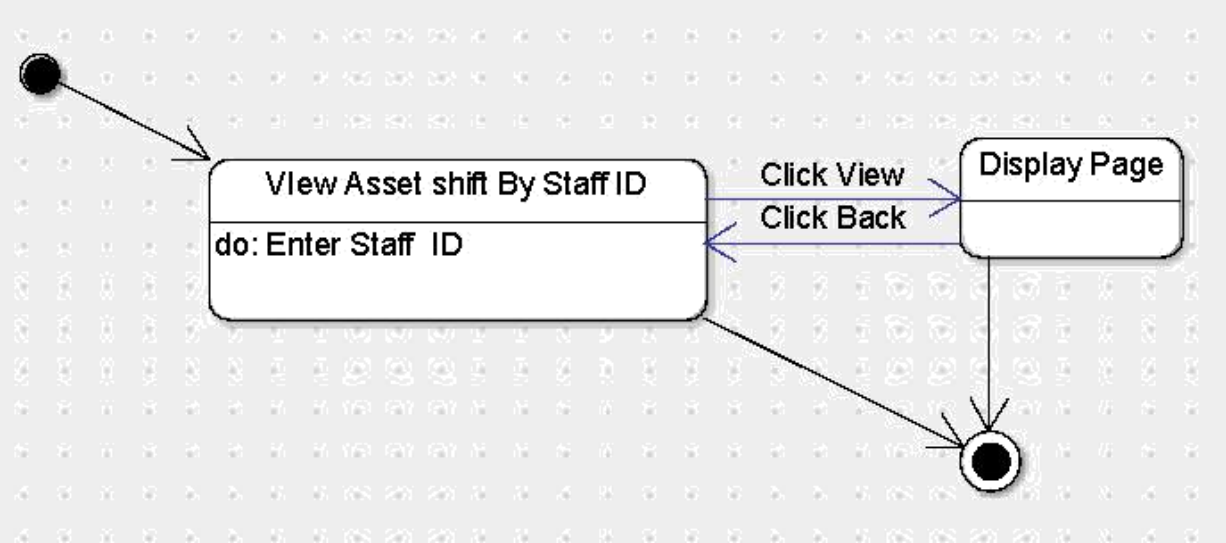
3.1.3.07 View Asset: The user gets the whole details and description i.e., status, location, date of purchase and many more just by providing the unique asset ID and clicking View button.



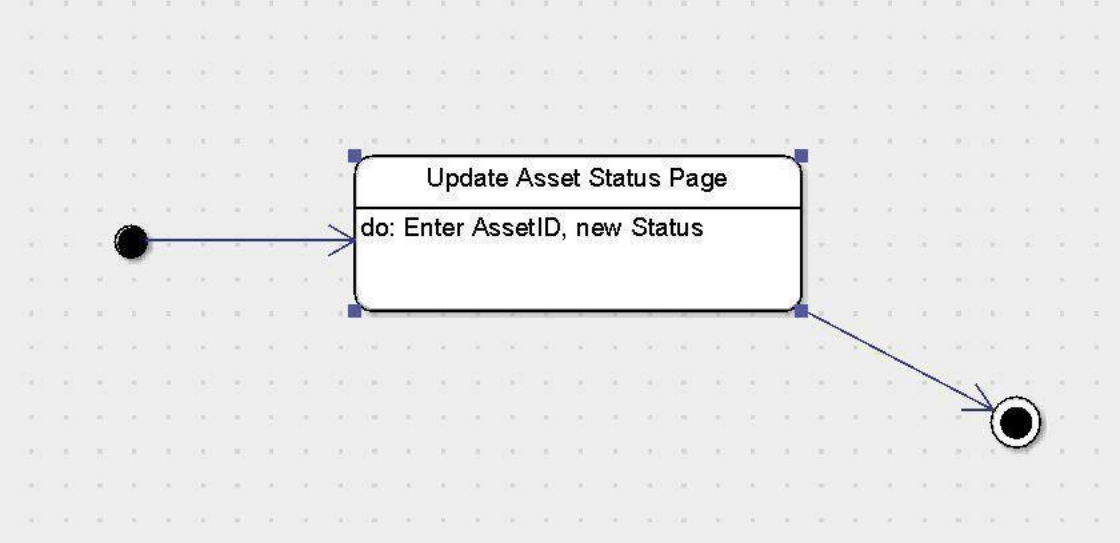
3.1.3.08 View Asset Shifting by Date: Here, the user provides from date and to date and gets all the shifting operations done in the duration just by clicking the View button.



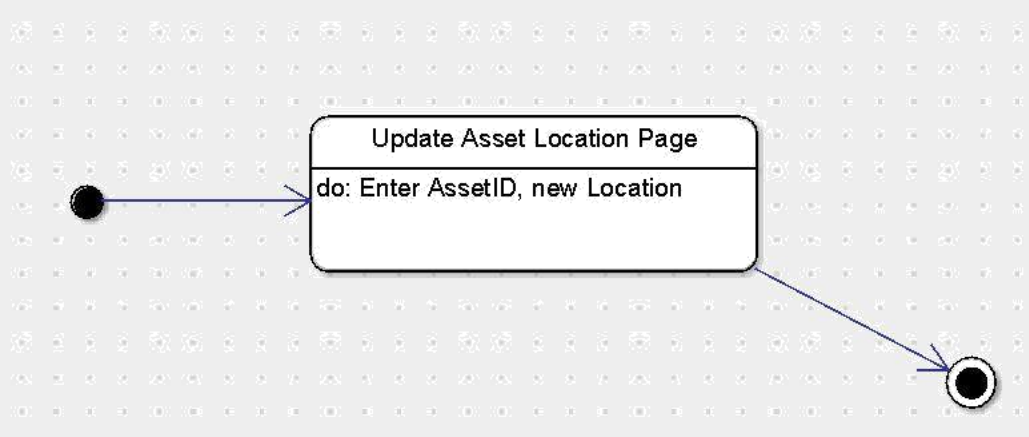
3.1.3.09 View Asset Shifting by Asset ID: Here, the user provides the Asset ID of an asset and gets all the shifting operations done to the asset from the time of purchase just by clicking the View button.



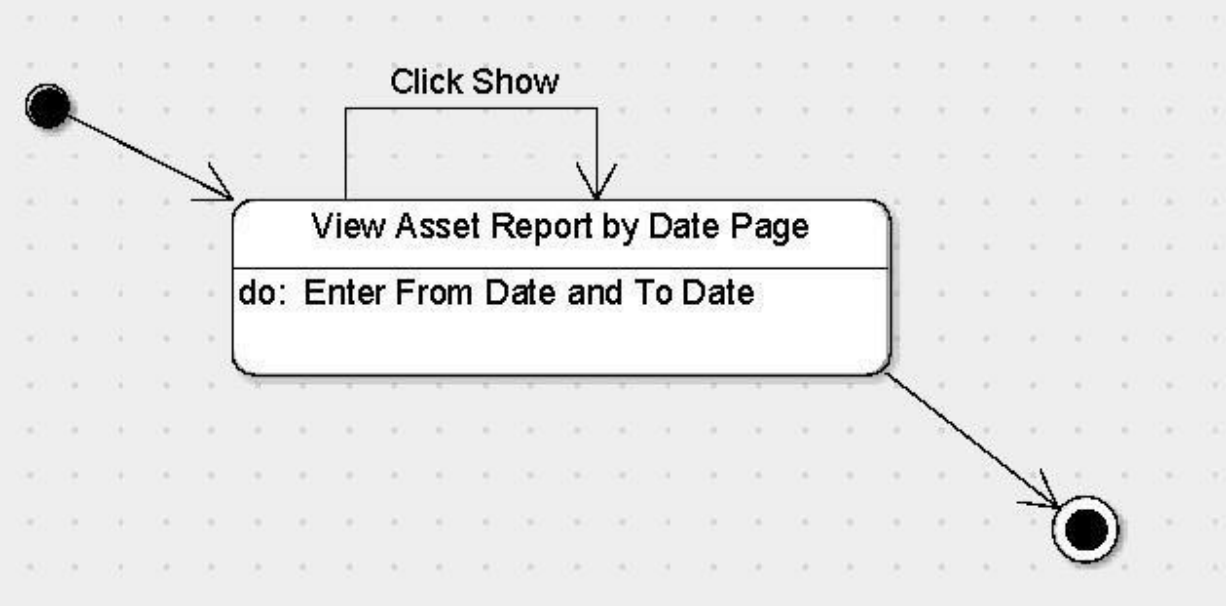
3.1.3.10 View Asset Shifting by Staff ID: Here, the user provides the Staff ID of a particular staff and gets all the shifting operations done by the staff from the time of joining the organization just by clicking the View button.



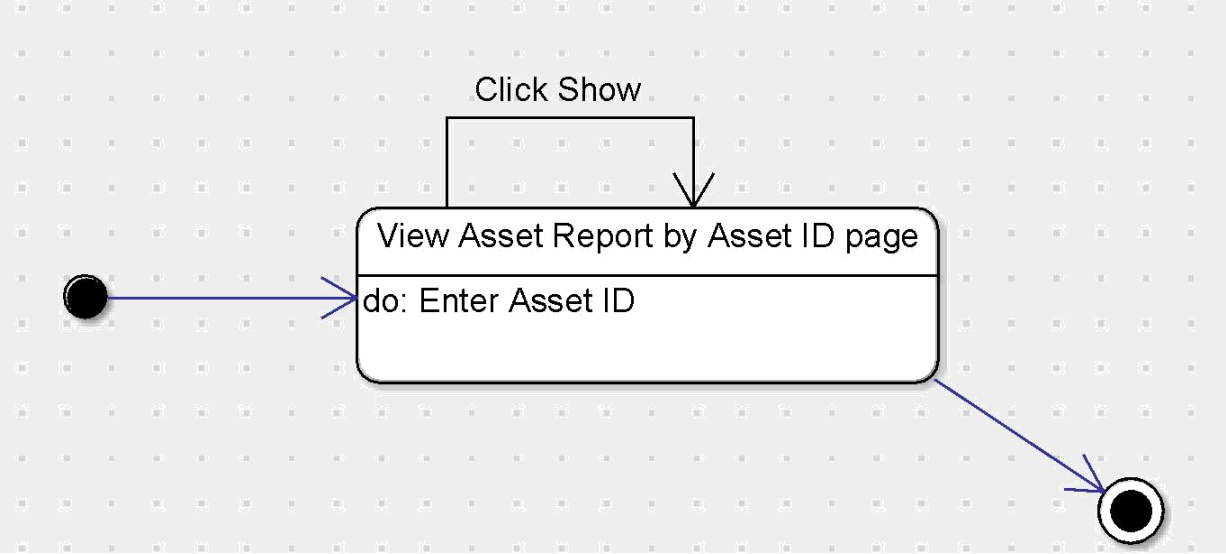
3.1.3.11 Update Asset Status: The user provides the asset ID of the asset and the new status which are predefined and by clicking the update button the status of the asset is updated.



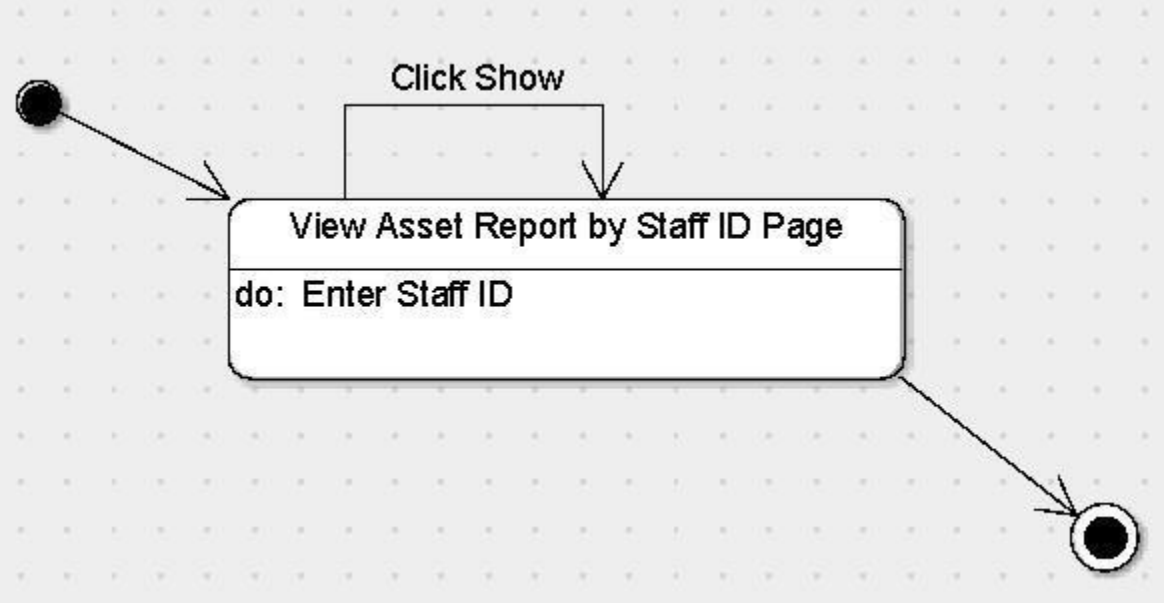
3.1.3.12 Update Asset Location: The user provides the asset ID of the asset, the new location and room to which the asset is being shifted to and by clicking the update button the location of the asset is updated.



3.1.3.13 View Report by Date: The user here gets the whole transactions occurring the whole-time frame between from date and to date. The output is a list of all the transactions in a tabular format in a different page.



3.1.3.14 View Report by Asset ID: The user here gets the whole transactions occurring to a single asset like adding, updating status, updating location and many more operations on a single asset just by providing the asset ID and clicking the Report button.



3.1.3.15 View Report by Staff ID: The user here gets the whole transactions done by a staff member like adding assets, updating asset status and locations and many other operations performed by a specific member of the team just by providing the staff ID and clicking the report button.

3.2Classname: HttpServlet

Description: This is an in-built class in java servlet package which is used in almost all the web-based projects built using java. It has a few basic functions which are can be used for easy programming and fewer code lines.

3.2.1 Method 1: doGet()

Input: HttpServletRequest request, HttpServletResponse response

Output:

void

Method Description:

Every action to be done passes through this function. The data which comes from the website from the form is stored in request object irrespective of the type and number of inputs. This method has response object which actually creates the response to the user in the form of dynamic web pages and puts it in a presentable format. All this is done in the doGet method and directing between web pages and .jsp files for appropriate functioning of the application.

3.2.2 Method 2: doPost()

Input: HttpServletRequest request, HttpServletResponse response

Output:

void

Method Description:

This function is generally used in the place of doGet() method in special cases where you prefer hiding the url of the server and the data you send from one page/jsp file to other page/jsp file.

3.2.3 Method 3: init()

Input: void

Output:

void

Method Description:

This is the servlet inbuilt function which actually builds the servlet. Extra functionality like starting the servlet with some objects and some other initial actions can be performed in this function.

3.2.4 Method 4: destroy()

Input: void

Output:

void

Method Description:

This is the servlet inbuilt function which runs at the end of any servlet. This function can print some message, free some heap area or some other actions which are to be performed at the end of the servlet. Like, the operations to be performed just before the servlet dies are to be written in this function.

3.3Class name: Action

Description: This is a servlet which inherits the HttpServlet class using the doGet() and doPost() methods of the parent class. This servlet is where the admin is headed after login. After login which includes authentication of the user, the user has a set of options to choose from. The user can then choose one of these just by hovering and clicking the option required. The servlet then forwards to the concerned class/jsp/servlet to perform the actions as per user's need.

3.3.1 Method 1: doGet()

Input: HttpServletRequest request, HttpServletResponse response

Output:

void

Method Description:

Every action to be done passes through this function. The data which comes from the website from the form is stored in request object irrespective of the type and number of inputs. This method has response object which actually creates the response to the user in the form of dynamic web pages and puts it in a presentable format. All this is done in the doGet method and directing between web pages and .jsp files for appropriate functioning of the application.

3.3.2 Method 2: doPost()

Input: HttpServletRequest request, HttpServletResponse response

Output: -

-

Method Description:

This function is generally used in the place of doGet() method in special cases where you prefer hiding the url of the server and the data you send from one page/jsp file to other page/jsp file.

3.4Classname:AuditorActionServ

Description: This is a servlet which inherits the HttpServlet class using the doGet() and doPost() methods of the parent class. This servlet is where the auditor is headed after login. After login which includes authentication of the user, the user has a set of options to choose from. The user can then choose one of these just by hovering and clicking the option required. The servlet then forwards to the concerned class/jsp/servlet to perform the actions as per user's need.

3.4.1 Method 1: doGet()

Input: HttpServletRequest request, HttpServletResponse response

Output:

void

Method Description:

Every action to be done passes through this function. The data which comes from the website from the form is stored in request object irrespective of the type and number of inputs. This method has response object which actually creates the response to the user in the form of dynamic web pages and puts it in a presentable format. All this is done in the doGet method and directing between web pages and .jsp files for appropriate functioning of the application.

3.4.2 Method 2: doPost()

Input: HttpServletRequest request, HttpServletResponse response

Output:

void

Method Description:

This function is generally used in the place of doGet() method in special cases where you prefer hiding the url of the server and the data you send from one page/jsp file to other page/jsp file.

3.5Classname: StaffActionServ

Description: This is a servlet which inherits the HttpServlet class using the doGet() and doPost() methods of the parent class. This servlet is where the staff member is headed after login. After login which includes authentication of the user, the user has a set of options to choose from. The user can then choose one of these just by hovering and clicking the option required. The servlet then forwards to the concerned class/jsp/servlet to perform the actions as per user's need.

3.5.1 Method 1: doGet()

Input: HttpServletRequest request, HttpServletResponse response
Output: void
Method Description:

Every action to be done passes through this function. The data which comes from the website from the form is stored in request object irrespective of the type and number of inputs. This method has response object which actually creates the response to the user in the form of dynamic web pages and puts it in a presentable format. All this is done in the doGet method and directing between web pages and .jsp files for appropriate functioning of the application.

3.5.2 Method 2: doPost()

Input: HttpServletRequest request, HttpServletResponse response
Output: void
Method Description:

This function is generally used in the place of doGet() method in special cases where you prefer hiding the url of the server and the data you send from one page/jsp file to other page/jsp file.

3.6 Class name: User

Description: This is a servlet which is a result of Action servlet created earlier.

3.6.1 Method 1: User()

Input: int id, String actor, String name
Output: void
Method Description:

This is the constructor of the class used to create the object of the class. The object of the class is created using this function with pre-defined parameters a user object can be created which can be easily inserted in database and actions of the user can also be defined.

3.7 Class name: AdminAction

Description: This is a java class which calls functions of other classes to perform each action. Every request from Action class gets distributed here and the whole application moves in a specific direction to perform a specific action.

3.7.1 Method 1: create()

Input: String name, String category

Output: Boolean created

Method Description:

This method adds the new type of asset created to the database. The assets to this type of asset are to be added using add asset function. This function just creates the type of the new asset.

3.7.2 Method 2:

AddAsset() Input: String productID **Output:** Boolean added **Method Description:**

This method actually adds the asset to the database where its type is already created. The method takes the input as product ID which is given to it by the Action servlet which is in its request object. It just gives a pop-up which tells the created assetID.

3.7.3 Method 3: AddMultAsset()

Input: String productid, int number

Output: Boolean added

Method Description:

This method actually adds multiple assets to the database where its type is already created. The method takes the input as product ID and number of assets to be added which is given to it by the Action servlet which is in its request object. It just gives a pop-up which tells the created assetIDs.

3.7.4 Method 4: show()

Input: String assetID

Output: String out

Method Description:

This method deals with 3 different actions. These three actions are performed by the methods in View class. This method just calls the concerned method of the class.

3.7.5 Method 5: updateStatus()

Input: String assetID, String stat
Output: Boolean out
Method Description:

This method takes in assetID and new status as input and updates the status of the object in the database.

3.7.6 Method 6:

updateLocation() Input: String assetID, String loc **Output:** Boolean out
Method Description:

This method takes in assetID and new location as input and updates the location of the object in the database.

3.7.7 Method 7: showAssetShifting()

Input: void
Output: void

Method Description: This method deals with 3 different actions. These three actions are performed by the methods in ViewAssetShifting class. This method just calls the concerned method of the class.

3.7.8 Method 8: showReport()

Input: void
Output: void
Method Description:

This method deals with 3 different actions. These three actions are performed by the methods in Report class. This method just calls the concerned method of the class.

3.8 Class name: AuditorAction

Description: This is a java class which calls functions of other classes to perform each action. Every request from Action class gets distributed here and the whole application moves in a specific direction to perform a specific action.

3.8.1 Method 1: show()

Input: String assetID
Output: String out
Method Description:

This method deals with 3 different actions. These three actions are performed by the methods in View class. This method just calls the concerned method of the class.

3.8.2 Method 2: showAssetShifting()

Input: void
Output: void
Method Description:

This method deals with 3 different actions. These three actions are performed by the methods in ViewAssetShifting class. This method just calls the concerned method of the class.

3.8.3 Method 3: showReport()

Input: void
Output: void
Method Description:

This method deals with 3 different actions. These three actions are performed by the methods in Report class. This method just calls the concerned method of the class.

3.8.4 Method 4: remove()

Input: String assetID
Output: Boolean out
Method Description:

When this method is called, all the records of the asset corresponding to the asset ID provided are deleted.

3.9Classname: StaffAction

Description: This is a java class which calls functions of other classes to perform each action. Every request from Action class gets distributed here and the whole application moves in a specific direction to perform a specific action.

3.9.1 Method 1: show()

Input: String assetID

Output: String out

Method Description:

This method deals with 3 different actions. These three actions are performed by the methods in View class. This method just calls the concerned method of the class.

3.9.2 Method 2: updateStatus()

Input: String assetID, String stat

Output: Boolean out

Method Description:

This method takes in assetID and new status as input and updates the status of the object in the database.

3.9.3 Method 3:

updateLocation() Input: String assetID, String loc **Output:** Boolean out

Method Description:

This method takes in assetID and new location as input and updates the location of the object in the database.

3.10 Class name: Asset

Description: This is a java class which creates an object for each asset object we deal with in each operation in order to produce faster results. This class's object is generally used to insert data into database and access any asset's data in the database easily and efficiently.

3.11Class name: ViewAssetShifting

Description: This is a java class which is use to perform 3 actions

- View Asset Shifting History by date
- View Asset Shifting History by Asset ID
- View Asset Shifting History by Staff ID

These three actions are performed by 3 different methods. The methods are called by AdminAction or AuditorAction or StaffAction class to perform the above-mentioned actions. These functions of the class are called by showAssetShifting() method of the above mentioned classes.

3.11.1 Method 1: byDate()

Input: String from, String to

Output:

void

Method Description:

This method takes in a from date and a to date and provides a history of all the Shifting operations done from the first date to the other. The output of this method is just a table of all the shifting history between those dates.

3.11.2 Method 2:

byAssetID() Input: String

assetID

Output:

void

Method Description:

This method takes in an assetID and provides a history of all the Shifting operations done to the asset whose assetID is provided. The output of this method is just a table of all the shifting history of an asset.

3.11.3 Method 3:

byStaffID() Input: String

staffID

Output:

void

Method Description:

This method takes in a staffID and provides a history of all the Shifting operations done by the staff member whose staffID is provided. The output of this method is just a table of all the shifting history done by a single staff member.

3.12Class name: Report

Description: This is a java class which is use to perform 3 actions

- View Comprehensive report by date
- View Comprehensive report by Asset ID
- View Comprehensive report by Staff ID

These three actions are performed by 3 different methods. The methods are called by AdminAction or AuditorAction or StaffAction class to perform the above-mentioned actions. These functions of the class are called by showReport() method of the above mentioned classes.

3.12.1 Method 1: byDate()

Input: String from, String to

Output:

void

Method Description:

This method takes in a from date and a to date and provides a history of all the operations done from the first date to the other. The output of this method is just a table of all the operations between those dates.

3.12.2 Method 2:

byAssetID() Input: String

assetID

Output:

void

Method Description:

This method takes in an assetID and provides a history of all the operations done to the asset whose assetID is provided. The output of this method is just a table of all the operations of an asset.

3.12.3 Method 3:

byStaffID() Input: String

staffID

Output:

void

Method Description:

This method takes in a staffID and provides a history of all the operations done by the staff member whose staffID is provided. The output of this method is just a table of all the operations done by a single staff member.

3.13 Class name: View

Description: This is a java class which is use to perform 3 actions

- View Status of an asset
- View Location of an asset
- View Asset

These three actions are performed by 3 different methods. The methods are called by AdminAction or AuditorAction or StaffAction class to perform the above-mentioned actions. These functions of the class are called by show() method of the above mentioned classes.

3.13.1 Method 1: viewAssetStatus()

Input: String
assetID

Output:
void

Method Description:

This method takes in an assetID and provides present status of the asset like, broken, can't be repaired, to be repaired, beyond economic rate and many more pre-defined states.

3.13.2 Method 2:

viewAssetLocation() Input: String
assetID

Output:
void

Method Description:

This method takes in an assetID and provides present location of the asset like, UG1, UG2, PG1, PG2, AC1, AC2 and many more pre-defined locations.

3.13.3 Method 3: viewAsset()

Input: String assetID

Output:
void

Method Description:

This method takes in an assetID and provides all the details about the asset like, category, name, date of purchase, status, location and other details.

4.0 Execution Architecture

Runtime environment required is any device supporting Java and an OS of Windows 7 or later versions.

The whole project was built using the MVC architecture in OOPs for a better code reusability as well as better understanding. Testing and development of code is easier and can be done any non-member of the group easily jus y reading this document as well as Software Requirements document provided earlier.

5.0 Design Decisions and Tradeoffs

The earlier design of the landing page had icons spread all over the page and the user could be directed to the concerned page. Thanks to JSP pages which can help us embed all the functionality easily in all the pages easily and with a simple use a tag. Now, the user can jump between pages easily just hovering on to the options seen at the top of every page and clicking it. Now, we have a comprehensive toolbar like headline for every page which can help the user to switch between actions and pages easily.

6.0 Pseudo Codes

Create():

1. Make connection with database
2. Sql1="Select prod_id from products;”
3. PreparedStatement ps=(PreparedStatement) con.prepareStatement(sql1);
4. ResultSet rs=ps.executeQuery();
5. id=rs.getString(1);
6. id=id+1;
7. Execute the following statement to store the new product details in database
8. Insert into products values(id,name,category);

ViewStatus():

1. Make connection with database
2. Take asset_id1 as input
3. sql1="Select status from assets where asset_id= asset_id1;”
4. rs = ps.executeQuery();
5. While rs.next() is TRUE repeat Step 6
6. status=rs.getString(1);

ViewLocation():

1. Make connection with database
2. Take asset_id1 as input
3. sql1="Select location and room from assets where asset_id= asset_id1;”
4. rs = ps.executeQuery();
5. While rs.next() is TRUE repeat Step 6
6. location=rs.getString(1)+" "+rs.getString(2);

ViewAsset():

1. Make connection with database
2. Take asset_id1 as input
3. sql1="Select * from assets where asset_id=asset_id1";
4. rs = ps.executeQuery();
5. While rs.next() is TRUE repeat Step 6
6. asset=rs.getString(1)+" "+rs.getString(2)+"
   "+rs.getString(3)+" "+rs.getString(4)+" "+rs.getString(5)+"
   "+rs.getString(6)+" "+rs.getTimestamp(7);

Remove():

1. Make connection with database
2. Take asset_id
3. Sql=”Remove assets where assetId=asset_id;”
4. Executing the Sql statement will remove the asset from database.

UpdateStatus():

1. Make connection with database
2. Take asset_id and status1 as inputs
3. Sql=”Update assets set status=status1 where assetId=asset_id;”
4. Sql=insert into status values(trans_id, prod_id,asset_id, status1,user_id, time)
5. Executing these two Sql statements will update the database.

UpdateLocation():

1. Make connection with database
2. Take asset_id , room1 and location1 as inputs
3. Sql=”Update assets set location=location1 and room=room1 where assetId=asset_id;”
4. Sql=insert into location values(trans_id, prod_id,asset_id, toLocation,toRoom, user_id, time)
5. Executing these two Sql statements will update the database.

ViewAssetShifting ByAssetID():

1. Make connection with database
2. Take asset_id1 as input
3. sql1="Select tolocation,toroom,time from location where asset_id=asset_id1;”
4. rs = ps.executeQuery();
5. While rs.next() is TRUE repeat Step 6
6. Print(rs.getString(1)+" - "+rs.getString(2)+" - "+rs.getString(3));

ByStaffID():

1. Make connection with database
2. Take Staff_id as input
3. sql1="Select prod_id,asset_id,tolocation,toroom,time from location where UserId=Staff_id;”
4. rs = PS.executeQuery();
5. While rs.next() is TRUE repeat Step 6
6. Print(rs.getString(1)+" - "+rs.getString(2)+" - "+rs.getString(3)+" - "+rs.getString(4)+" - "+rs.getString(5));

ReportByDate():

ReportByAssetId( ):

1. Make connection with database
2. Take asset_id1 as input
3. sql1="Select tolocation,toroom,time from location where asset_id=asset_id1;”
4. rs = ps.executeQuery();
5. Print(“ Shifting”);
6. While rs.next() is TRUE repeat Step 7
7. Print(rs.getString(1)+" - "+rs.getString(2)+" - "+rs.getString(3));
8. sql1="Select status,time from status where asset_id=?;”
9. rs = ps.executeQuery();
10. Print(“ Status”);
11. While rs.next() is TRUE repeat Step 12
12. Print(rs.getString(1)+" - "+rs.getString(2))

ReportByStaffId():

1. Make connection with database
2. Take staff_id1 as input
3. sql1="Select tolocation,toroom,time from location where asset_id=staff_id1;”
4. rs = ps.executeQuery();
5. Print(“ Shifting”);
6. While rs.next() is TRUE repeat Step 7
7. Print(rs.getString(1)+" - "+rs.getString(2)+" - "+rs.getString(3));
8. sql1="Select status,time from status where staff_id=?;”
9. rs = ps.executeQuery();
10. Print(“ Status”);
11. While rs.next() is TRUE repeat Step 12
12. Print(rs.getString(1)+" - "+rs.getString(2));

Logout():

1. Get session object from request
2. Session.invalidate();

Login():

1. Make connection with database
2. Take user_name and password as input
3. sql="Select * from UserN where UserID=user_name and password=aes_encrypt(password,key);”
4. rs = ps.executeQuery();
5. While rs.next() is TRUE Repeat Steps 6 to Step 11
6. actor=rs.getString(2);
7. name=rs.getString(3);
8. if(actor==null)
9. Print(“username or password is wrong”);
10. Else
11. Print(“Successful login”);

CHAPTER 3
(METRICS)

Description

The following metrics were calculated for each class:

- WMC: Weighted methods per class
- DIT: Depth of Inheritance Tree
- NOC: Number of Children
- CBO: Coupling between object classes
- RFC: Response for a Class
- LCOM: Lack of cohesion in methods
- Ca: Afferent coupling (not a C&K metric)
- NPM: Number of Public Methods for a class (not a C&K metric)

The corresponding metrics for each class: WMC, DIT, NOC, CBO, RFC, LCOM, Ce, and NPM.

Metrics (for each class)

Action Class

filename
C:\Users\CHAITANYA\workspace\AssetManagement\build\classes\controller\
Action.class
compiled from Action.java
compiler version 52.0
access flags 33
constant pool 291 entries
ACC_SUPER flag true
Attribute(s):
SourceFile(Action.java)
(Unknown attribute RuntimeVisibleAnnotations: 00 01 01 20 00 01 01 21 5b 00...
(truncated))
1 fields:
3 methods:
public void <init>()
protected void doGet(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException

protected void doPost(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException

controller.Action 3 0 0 2 30 3 0 1

AdminAction Class

filename
C:\Users\CHAITANYA\workspace\AssetManagement\build\classes\controller\
AdminAction.class
compiled from AdminAction.java
compiler version 52.0
access flags 33
constant pool 234 entries
ACC_SUPER flag true
Attribute(s):
SourceFile(AdminAction.java)
6 methods:
public void <init>(int id, String name, String actor)
String create(String name, String category)
String addasset(String name)
String add(String name, int count)
String updateStatus(String asset_id, String status)
String updateLocation(String asset_id, String location, String
room)

controller.AdminAction 6 0 0 4 32 15 0 1

AuditorActionClass

filename
C:\Users\CHAITANYA\workspace\AssetManagement\build\classes\controller\
AuditorAction.class

compiled from AuditorAction.java
compiler version 52.0
access flags 33
constant pool 248 entries
ACC_SUPER flag true
Attribute(s):
SourceFile(AuditorAction.java)
1 fields:
String function
6 methods:
public void <init>(int id, String name, String actor)
String addasset(String name)
String add(String name, int count)
String updateStatus(String asset_id, String status)
String updateLocation(String asset_id, String location, String room)
String remove(String asset_id)

controller.AuditorAction 6 0 0 4 32 15 0 1

AuditorActionServ Class

filename
C:\Users\CHAITANYA\workspace\AssetManagement\build\classes\controller\
AuditorActionServ.class
compiled from AuditorActionServ.java
compiler version 52.0
access flags 33
constant pool 229 entries
ACC_SUPER flag true
Attribute(s):
SourceFile(AuditorActionServ.java)
(Unknown attribute RuntimeVisibleAnnotations: 00 01 00 e2 00 01 00 e3 5b 00...
(truncated))
1 fields:
3 methods:
public void <init>()
protected void doGet(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException
protected void doPost(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException

controller.AuditorActionServ 3 0 0 2 26 3 0 1

Main Class

filename
C:\Users\CHAITANYA\workspace\AssetManagement\build\classes\controller\
main.class
compiled from main.java
compiler version 52.0
access flags 33
constant pool 199 entries
ACC_SUPER flag true

Attribute(s):
SourceFile(main.java)
(Unknown attribute RuntimeVisibleAnnotations: 00 01 00 c4 00 01 00 c5 5b 00...
(truncated))
4 fields:
private static final long serialVersionUID = 1
static final String url = "jdbc:mysql://localhost:3306/asset"
static final String user = "root"
static final String pass = "lokesh1999"
3 methods:
public void <init>()
protected void doGet(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException
protected void doPost(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException

controller.main 3 0 0 4 25 3 0 1

StaffAction Class

filename
C:\Users\CHAITANYA\workspace\AssetManagement\build\classes\controller\
StaffAction.class
compiled from StaffAction.java
compiler version 52.0
access flags 33

constant pool 203 entries
ACC_SUPER flag true
Attribute(s):
SourceFile(StaffAction.java)
1 fields:
String function
3 methods:
public void <init>(int id, String name, String actor)

String
room)

controller.StaffAction 3 0 0 4 28 3 0 1

StaffActionServ Class

filename C:\Users\CHAITANYA\workspace\AssetManagement\build\classes\controller\StaffActionServ.class
compiled from StaffActionServ.java
compiler version 52.0
access flags 33
constant pool 198 entries
ACC_SUPER flag true
Attribute(s):
SourceFile(StaffActionServ.java)
(Unknown attribute RuntimeVisibleAnnotations: 00 01 00 c3 00 01 00 c4 5b 00... (truncated))
1 fields:
3 methods:
public void <init>()
protected void doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException, java.io.IOException
protected void doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException, java.io.IOException

controller.StaffActionServ 3 0 0 2 24 3 0 1

Time Class

controller.time 2 1 0 0 6 1 0 2

View Class

controller.View 4 1 0 2 19 6 0 1

User Class

model.user 7 1 0 0 8 3 0 7

MainView Class

filename C:\Users\CHAITANYA\workspace\AssetManagement\build\classes\view\mainView.class
compiled from mainView.java
compiler version 52.0
access flags 33
constant pool 138 entries
ACC_SUPER flag true

Attribute(s):
SourceFile(mainView.java)
(Unknown attribute RuntimeVisibleAnnotations: 00 01 00 87 00 01 00 88 5b 00... (truncated))
1 fields:
3 methods:
public void <init>()
protected void doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException, java.io.IOException
protected void doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException, java.io.IOException

view.mainView 3 0 0 1 16 3 0 1

Appendix

Here is the description of each metric calculated,

WMC - Weighted methods per class

A class's *weighted methods per class* WMC metric is simply the sum of the complexities of its methods. As a measure of complexity, we can use the cyclomatic complexity, or we can arbitrarily assign a complexity value of 1 to each method. The value of the WMC is equal to the number of methods in the class.

DIT - Depth of Inheritance Tree
The *depth of inheritance tree* (DIT) metric provides for each class a measure of the inheritance levels from the object hierarchy top. In Java where all classes inherit Object the minimum value of DIT is 1.

NOC - Number of Children

A class's *number of children* (NOC) metric simply measures the number of immediate descendants of the class.

CBO - Coupling between object classes

The *coupling between object classes* (CBO) metric represents the number of classes coupled to a given class (efferent couplings, Ce). This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions.

RFC - Response for a Class

The metric called the *response for a class* (RFC) measures the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object). Ideally, we would want to find for each method of the class, the methods that class will call, and repeat this for each called method, calculating what is called the *transitive closure* of the method's call graph. This process can however be both expensive and quite inaccurate. In *ckjm*, we calculate a rough approximation to the response set by simply inspecting method calls within the class's method bodies. This simplification was also used in the 1994 Chidamber and Kemerer description of the metrics.

LCOM - Lack of cohesion in methods

A class's *lack of cohesion in methods* (LCOM) metric counts the sets of methods in a class that are not related through the sharing of some of the class's fields. The original definition of this metric (which is the one used in *ckjm*) considers all pairs of a class's methods. In some of these pairs both methods access at least one common field of the class, while in other pairs the two methods to not share any common field accesses. The lack of cohesion in methods is then calculated by subtracting from the number of method pairs that don't share a field access the number of method pairs that do. Note that subsequent definitions of this metric used as a measurement basis the number of disjoint graph components of the class's methods. Others modified the definition of connectedness to include calls between the methods of the class. The program *ckjm* follows the original (1994) definition by Chidamber and Kemerer.

Ca - Afferent couplings

A class's afferent couplings is a measure of how many other classes use the specific class. Ca is calculated using the same definition as that used for calculating CBO (Ce).

NPM - Number of Public Methods

The NPM metric simply counts all the methods in a class that are declared as public. It can be used to measure the size of an API provided by a package.

CHAPTER 4

(MoM 1&2)

Meeting Minutes

Project Group

Address- NIIT University, Neemrana,

Date -21/8/2018

Delhi-Jaipur Highway, Rajasthan

301705, Contact-9000292903, 8349104422

Topic- Meeting with Administration Head of NIIT University about Project Idea of Hostel Asset Management System

Attendees:

Name	Designation
Nitin Gupta	Administration Head
Vamsi PSK	Developer
Pisupati Chaitanya	Developer
Pradeep Yadav	Developer
Mamidi Lokesh	Developer
Jobin Joseph	Developer

Absentees:

Name	Designation	Reason
NONE	N/A	N/A

Agenda at hand − Meeting with Nitin Gupta and discussing with him about feasibility and applicability of shifting paper-based tracking and planning of Asset Management to an automated system.

Issues raised – Record handling and control, asset operations.

Suggestions:

Name	Suggestions
Nitin Gupta	Providing assets with asset IDs
Vamsi PSK	Hierarchy of access using Chain of Command
Pisupati Chaitanya	Actions attached with each actor
Pradeep Yadav	Manual Supervision over the system
Mamidi Lokesh	Asset identity and segregation of variety
Jobin Joseph	Security and throughput aspects

Decision – Preparing SRS and providing of the raw data by admin office.

Future Meetings – 21st September, 2018 at 12:30 PM at location TBD.

Issuer: Jobin Joseph **Date of approval:** 28/8/2018

Meeting Minutes

Project Group

Address- NIIT University, Neemrana, Date -21/9/2018

Delhi-Jaipur Highway, Rajasthan

301705, Contact-9000292903, 8349104422

Topic- Meeting with Administration Head of NIIT University about proceeding with Hostel Asset Management System project after having developed Software Requirement Specification Document.

Attendees:

Name	Designation
Nitin Gupta	Administration Head
Vamsi PSK	Developer
Pisupati Chaitanya	Developer
Pradeep Yadav	Developer
Mamidi Lokesh	Developer
Jobin Joseph	Developer

Absentees:

Name	Designation	Reason
NONE	N/A	N/A

Agenda at hand − Meeting with Nitin Gupta and discussing with him about requirements and resources needed that act as a backbone to the software under-development and understanding the flow structure for the design of the software.

Issues raised – Obtaining the optimal design, maximizing throughput, secure design to make it applicable in accordance with the current system undertaken by NIIT University, use of MVC.

Suggestions:

Name	Suggestions
Nitin Gupta	Incorporating the theme of flow already in use by NIIT University
Vamsi PSK	MVC distribution
Pisupati Chaitanya	Throughput and optimal design
Pradeep Yadav	Design oriented technical amendments
Mamidi Lokesh	Foolproofing the flow of design
Jobin Joseph	Security-in-depth aspect

Decision – Preparing SDS and optimizing the output of productive work intake.

Future Meetings – 21st October, 2018 at 12:30 PM at Location TBD.

Issuer: Jobin Joseph **Date of approval:** 28/9/2018

CHAPTER 5
(TESTING)

Requirement ID	Class name	Method name	Input Parameter	Expected Output	Real Output	Status(Fail/Pass)
2	Action	Create	"laptop","electronic"	"P104"	"P104"	Pass
			"laptop","electronic"	"Product Already Exist"	"Product Already Exist"	Pass
3	Action	Addasset	"P104"	"A10011 Asset Added"	"A10011 Asset Added"	Pass
4	Action	Add	"P104", 2	"A10012 to A10013 Asset Added"	"A10012 to A10013 Asset Added"	Pass
16	Action	UpdateStatus	"A10001","damaged"	"Updated"	"Updated"	Pass
15	Action	UpdateLocation	"A10001","UG1","1304"	"Updated"	"Updated"	Pass
5	View	ViewStatus	"A10003"	"working"	"working"	Pass
			"A101"	""	""	Pass
6		ViewLocation	"A10003"	"AC1 rec"	"AC1 rec"	Pass
			"A1001"	""	""	Pass
7		ViewAsset	"A10003"	"A10003 P100 AC1 rec 00001 2018-11-15 20:24:10.0"	"A10003 P100 AC1 rec 00001 2018-11-15 20:24:10.0"	Pass
			"A1004"	""	""	Pass

CHAPTER 6

(Nature of the Customer)

Nature of the customer can be described as follows-

- Able to understand the demands and Supplements when it comes to asset requirements on a timely basis and otherwise.
- Administrator staff of said institution or holding.
- Familiar to the concepts of digital platforms and related macro-technologies.
- Able to report changes and oversee the categorised work associated with the respective actor body.

CHAPTER 7

(Tools and Technologies Used During the Project Development)

Tools and technologies used during the project development have been divided according to steps of development-

- SRS
  - Microsoft Word Document
  - ArgoUML
    - Class Diagram
- SDS
  - Microsoft Word Document
  - LucidCharts
    - Flow Architecture
  - ArgoUML
    - State Diagram
    - Sequence Diagram
    - Class diagram
- Metrics
  - CKJM (Chidamber and Kemerer Java Metrics)
- Development
  - Eclipse- Photon
    - Java
    - Bootstrap
    - HTML
    - CSS
    - XML
    - Eclipse Debugger
  - MySQL Workbench
    - SQL
- Testing
  - Junit Test

CHAPTER 8
(Novelty of the Project Idea)

The idea of an asset management system is the child of the necessity felt by us and the NU administration, the necessity of having an automated system of record keeping and maintaining with the least manual work induced in its system and work-flow, also proving productive with the amount of time and load of work that is orchestrated with the given parameters.

Although our project is not patent-demanding but we have combined some existing technologies and formulated a way to extract the essence of asset management with respect to the conventions followed by NU and have created a standalone solution to the concerning problem statement hence creating a service which is as novel as it gets when it comes to automation of an existing service.

CHAPTER 9

(Sophistication Value of the Project)

The software is considered to be sophisticated enough for multiple reasons. This specific software deals with JSP pages, html and CSS formats with servlets creating these pages dynamically for each step. All the back-end modelling part is done in Java and a specific architecture or model is used. This model is termed as MVC where the code reusability is maximum with a faster rate of processing.

The sophistication level of user interface and backend of the project is vastly less when compared to the manual job associated to the same task and it is vastly proficient when utilizing human resources as well.

CHAPTER 10

(Applicability of the Project)

This specific software is going to be an intangible asset for the university in coming years. The software can help the end customer with all the asset management issues faced in day to day scenario.