12-2017

Securing Online Reputation Systems Through Temporal and Trust Analysis

Sai Shruthi Veerannagari
St. Cloud State University, sveerannagari@stcloudstate.edu

Securing Online Reputation Systems Through Temporal and Trust Analysis

by

Sai Shruthi Veerannagari

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

December, 2017

Starred Paper Committee:
Dennis Guster, Chairperson
Lynn A. Collen
Kasi Balasubramanian

Abstract

With the rapid advancement of reputation systems in different online social networks manipulations against these systems are developing rapidly. In this paper, TATA, the abbreviation of joint Temporal and Trust Analysis, will be described. TATA protects reputation systems from the combination of time domain anomaly detection and Dempster–Shafer hypothesis based trust calculation. Real user attack information gathered from a cyber-competition is then utilized to develop the testing data set. TATA accomplishes a significantly better result in determining the underlying things under attack, as well as detecting malicious users who insert dishonest ratings and recovering reputation scores.

Acknowledgement

The successful completion of this paper could not have been possible without the guidance of my beloved professors, Dr. Dennis Guster and Professor Lynn A. Collen. I also would like to thank Professor Kasi Balasubramanian for being part of the committee and finding the time to read my thesis.

I also would like to thank my mother V Vasantha, my father V H Laxma Reddy, and my friends who provided me immense support the entire way.

Table of Contents

Page

Chapter Page

Chapter Page

Chapter Page

List of Figures

Chapter I: Introduction

Introduction

Online reputation systems are playing a progressively critical role in impacting individuals' online purchasing/downloading decisions. Meanwhile manipulations against these systems, which excessively expand or empty reputation scores of online things, are also developing rapidly.

A Dempster-Shafer hypothesis based trust model is proposed to identify malicious users who embed unfair ratings to mislead items' reputation scores. In particular, the author explored time domain rating data for analyzing user behavior anomalies and, in view of this, the development of Dempster-Shafer hypothesis based trust model to further identify malicious users. When tested against real user attack data, information gathered from a cyber-competition, it exhibited a good performance in identifying malicious users.

Problem Statement

The main problem with online reputation systems is due to the anonymity of the Internet, it is extremely difficult for typical users to assess a stranger's reliability and quality, which then makes online interactions inherently risky.

Nature and Significance of the Problem

The issue, then, becomes the way which online participants secure themselves by judging the nature of strangers or unfamiliar items before making a decision. To address this issue, online reputation systems have been developed to aid in the decision-making process. The goal is to make large-scale virtual word of-mouth networks where people share opinions and experiences. They can create reviews and ratings on different things, including items such as services, digital content, and even on other individuals. These opinions and experiences, which are called user feedback, are gathered as evidence and are investigated,

aggregated, and disseminated to the other users of the system. The disseminated results are called a reputation score. Such systems are also referred to as feedback based reputation systems.

A reputation system defense plan is proposed, named TATA, for feedback-based reputation systems. TATA is the abbreviation of joint Temporal and Trust Analysis. It contains two modules—a time domain anomaly detector and a trust model in view of the Dempster-Shafer hypothesis. In particular, the ratings for a given item are considered as a time sequence, and a time domain anomaly detector is used to distinguish suspicious time intervals where anomaly occurs. A trust analysis is then used on the anomaly detection results. The idea of user behavior uncertainty is then taken from the Dempster-Shafer hypothesis to model users' behavior patterns, and to assess whether a user's rating value for each item is reliable or not.

Objective of the Study

The objective of this study is to demonstrate the incredible potential to effectively remove dishonest ratings and keep the online reputation system as secure and effective as possible for the online marketplace.

Definition of Terms

The Dempster-Shafer theory: A framework for joining evidence from distinct sources to accomplish a level of belief.

> Consider two events, where a = good behavior and b = bad behavior, and a subject is observed to perform good behaviors for 'r' times and perform bad behaviors for 's' times.
>
> Bg = r/r+s+2,
>
> Bb = s/r+s+2,

$$U = 2/r+s+2,$$

Where Bg is the belief that the proposition that the subject will perform a good behavior is genuine, Bb is the belief that the proposition that the subject will perform a bad behavior is genuine, and u is the uncertainty.

Behavior value: A user's behavior value is defined on a single item as a binary value to indicate whether his/her rating behavior is good or bad.

Combined binary value: The combined behavior value is introduced to evaluate user's behavior on multiple items.

Behavior uncertainty: A user's behavior uncertainty is defined using the Dempster-Shafer theory.

Summary

This chapter provided a brief introduction, the problem statement, and the objective of the study. It also provided a definition of the terms used.

Chapter II: Background and Review of Literature

Introduction

In this chapter, the related work and literature related to the problem and methodology will be covered.

Background Related to the Problem

Many individuals are currently involved in the manipulation of online reputation systems. Therefore, there is a need to create a defensive plan, and the securing of reputation systems has developed accordingly. The efforts to provide a defense are divided into four classifications. In the first class, the defense method used is to restrain the maximum number of ratings every user can give within a certain period of time. This sort of method actually confines the rating force of every user ID. This can then be used to stop the attackers from embedding a lot of dishonest ratings through a couple user IDs within a brief period of time.

In the second class, the defense plans intend to expand the cost of launching an attack. Some reputation systems, such as the one used by Amazon, allot a higher weight to users who rate verified transactions. This system then expands the cost to manipulate a competitor's items reputation value. However, it has little effect on attacks in which the attackers purchase their own particular items for reputational boosting. Some different schemes expand the costs of securing different user IDs by tying identities with a specific IP address or utilizing system coordinates to identify Sybil assaults. Such methods will extraordinarily increase the cost of the attack, however, it can't defeat attackers who have a lot of assets and determination. For instance, some organizations who work to boost reputations will regularly obtain a vast affiliate network of user IDs to try and thwart the defenses placed against them.

In the third class, the defense methods explore rating statistics. They consider ratings as arbitrary variables and accept that dishonest ratings have statistical appropriations different

from expected ratings. Representative plans are as per the following: A Beta function based methodology expects that the hidden ratings follow the Beta distribution and considers the ratings outside (lower) and (upper) quantile of the majority's opinions as dishonest ratings. An entropy-based method recognizes the ratings that acquire a critical change in the instability of the rating distribution as dishonest ratings. Dishonest rating analysis is based on a Bayesian model. Controlled anonymity and cluster separating are then utilized to wipe out dishonest ratings.

The defense approaches in the fourth class examine users' rating practices. Accepting that users with a bad rating history tend to give dishonest ratings, such methods decides the weight of a rating based on the notoriety or reputation of the user who gives the rating. Such reputation value is also referred to as trust or reliability. A few representative plans are as follows: The Iteration refinement method proposes assigned weights to a user's rating as per the reverse of this user's rating variance. A customized trust structure is presented so that different users may assign different trust values to the same user. A user's trust is obtained by accumulating other users' beliefs through the use of a belief theory. The REGRET reputation system calculates the user's reputation based on fuzzy logic. Flow models for example, the Eigen Trust and Google Page Rank, compute trust or reputation by transitive emphasis through circled or arbitrarily long chains.

Protecting online rating systems from unfair ratings. Online rating systems have been used by online exchange groups to boycott "awful" specialist co-ops and invite them to give "great" services. The execution of the internet rating systems is effectively bargained by different unjustifiable evaluations, e.g., balloting, castigating, and reciprocal unjustifiable ratings. Step-by-step instructions to alleviate the impact of the out of line evaluations remains an essential issue in internet rating frameworks. In this paper, a novel entropy-based strategy

is proposed to quantify the rating quality and in addition to screen the unjustifiable ratings. Test results demonstrate that the proposed technique is both viable and effective in lightening the impact of various sorts of unfair ratings.

Filtering out unfair ratings in Bayesian reputation systems. The nature of a reputation system relies on the integrity of the ratings it gets from its users. A basic issue is that a rater can rate a specialist more positively or more negatively than the genuine interaction with the operator would have justified. At the point when ratings are given by specialists outside the control of the defending party, it is almost impossible to know when a rater gives such uncalled-for ratings. It is frequently the case that unreasonable ratings have a different measurable example than reasonable ratings. This paper utilizes that thought, and depicts a factual method for barring unjustified ratings, and illustrates its adequacy through reenactments.

Information filtering via iterative refinement. With the explosive development of accessible data, especially on the Internet, assessment based filtering has turned into a vital assignment. Different systems have been devised with plans to deal with huge volumes of data and select what is probably going to be more applicable to a given situation. In this way, the user examines another positioning strategy, where the reputation of data suppliers is resolved self-reliably.

Reputation Trap: A powerful attack on reputation system offline sharing p2p environment. As the negative perception of reputation systems is broadly perceived, the motivation to control such systems is rapidly developing. TAUCA, a plan that distinguishes malicious clients and recuperates reputation scores from a novel point—a mix of temporal investigation and client connection analysis. Aided by the rich data in the time-area, TAUCA recognizes the items under assault, the time when the assaults occur, and the malicious clients

who embed unfair ratings. TAUCA and two other agent plans are tested against genuine client assault information gathered through a digital competition. TAUCA shows some significant advantages, it enhances the discovery rate and diminishes the false caution rate in the recognition of malicious clients. It also successfully diminishes the bias in the recovered reputation scores.

Literature Related to the Problem

Manifestations of trust are easy to perceive because users encounter and depend on them constantly, however, trust is very challenging to define because it shows itself in a wide range of forms. The literature on trust can also be very confusing because the term is being utilized with a variety of implications (McKnight & Chervany, 1996). Two common definitions of trust are called reliability trust and decision trust and are described the study. As the name suggests, reliability trust can be interpreted as the reliability of something or somebody, and the definition by Gambetta (1990) gives an example of how this can be determined: “Definition 1 (Reliability Trust): Trust is the subjective probability, by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends” (¶ 2).

This definition incorporates the idea of depending on the trusted party, and the reliability (likelihood) of the trusted party, as seen by the trusting party.

However, the trust can be more complex than Gambetta’s (1990) definition demonstrates. For instance, Falcone and Castelfranchi (2001) perceived that having high (reliability) trust in an individual, in general, is not enough to choose to enter into a circumstance of depending on that individual. Falcone and Castelfranchi wrote:

> For example, it is possible that the value of the damage per se (in case of failure) is too high to choose a given decision branch, and this independently either from the

> probability of the failure (even if it is very low) or from the possible payoff (even if it is very high). In other words, that danger might seem to the agent an intolerable risk. (p. 56)

In order to catch this expansive idea of trust, the following definition inspired by McKnight and Chervany (1996) can be utilized. “Definition 2 (Decision Trust): Trust is the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible.” The relative ambiguity of this definition is valuable because it makes it broader. It both explicitly and implicitly incorporates parts of a wild idea of trust which is dependent on the trusted element or party. The reliability of the trusted element or party is taken into account, as well as the utility as in positive utility will result because of a positive outcome, and that negative utility will result because of a negative outcome. Last, a certain risk attitude in the trusting party is created to acknowledge the situational risk resulting from the past components.

The idea of reputation is firmly connected to that of reliability, yet it is obvious that there is a clear and vital difference. For the purpose of this study, the reputation is defined according to the Concise Oxford lexicon. “Definition 3 (Reputation): Reputation is what is generally said or believed about a person or thing’s character or standing” (Reputation, n.d.). This definition relates well with the perspective of social system analysts (Freeman, 1979; Marsden & Lin, 1982) that reputation is a quantity derived from the hidden social organization which is globally visible to all individuals from the system.

According to Resnick, Zeckhauser, Friedman, and Kuwabara (2000), reputation systems must have the following three properties to operate at all:

- Entities must be long-lived, so that with every interaction there is always an expectation of future interactions.
- Ratings about current interactions are captured and distributed.
- Ratings about past interactions must guide decisions about current interactions.

Literature Related to the Methodology

There are a variety of existing models available for securing the Reputation system from diverse techniques of attack. To begin with, numerous ratings given by every user within a certain period of time (Liu & Sun, 2010). Second, the same user ID giving the rating for the same item for multiple times. Also, there could be various fake identities available and this was considered by the Sybil attacks (Yu, Kaminsky, Gibbons, & Flasman, 2006). Third, investigating users' rating behaviors by assigning the trust value to every user. If the user has a higher trust value, they are considered to be a good user and users having a lower trust value are considered to be a malicious user as identified by the Trust Evaluation (Whitby, Josang, & Indulska, 2005). Fourth, investigating rating distributions by using correlation techniques (Liu & Sun, 2010). After this, the malicious user is identified and also the exact score of the specific item is identified.

Summary

This chapter covered the background related work of the study and also discussed the literature of the problem and methodology.

Chapter III: Methodology

Introduction

In this chapter, the framework of the study, tools, and techniques that will be used are discussed. And also, the procedure for data collection is given in detail.

Design of the Study

A reputation defense plan is proposed, named TATA, for feedback-based reputation systems. Here, TATA is the abbreviation of joint Temporal and Trust Analysis. It contains two modules—a time domain anomaly detector and a trust model in view of the Dempster–Shafer hypothesis. In particular, the ratings to a given item are considered as a time sequence, and a time domain anomaly detector is acquainted with distinguishing suspicious time intervals where an anomaly happens. A trust analysis is then led based on the anomaly detection results. The idea of user behavior uncertainty is from the Dempster–Shafer hypothesis to model users' behavior patterns, and assess whether a user's rating value to each item is reliable or not.

A quantitative approach is used in this study. As the group studied here is larger and randomly selected, and the types of data collected are numbers and statistics and allows for the identification of statistical relationships.

Data Collection

The attackers are identified by using the procedures of Trust Evaluation and User Correlation.

General description. The proposed method contains two procedures: Trust Evaluation and User Correlation. In this paper, the detection of an attacker from distinctive points is accomplished and also a change detector is used to recognize the attacker within a specific time interval. But sometimes, in this system, some normal users may give rating

values too high or too low because of their unusual experience. So, a typical user could be considered as an attacker erroneously. The Trust Evaluation system distinguishes the attacker by the trust value created and, if the user has a trust value over the threshold value, they are considered to be a good client. The user having a trust value beneath the threshold value is at that point considered a malicious client. Subsequent to distinguishing the malicious user, the user correlation can be calculated by utilizing the correlation algorithm. Finally, the malicious user is identified and the ratings to the identified item are removed. The remaining ratings are then used to compute the product reputation.

Changing in rating sequence. In this example, the rating sequence can be given by the user who buys their item through an online purchase. Here the malicious users can be recognized by the user giving the rating within a specific time interval. For instance, if one user gives a rating at time 11.00 and then the same user gives further rating values at time 11.01, 11.02, and so on. Then that user is considered to be a malicious user. Specifically, in a time interval, normal users also give rating values. In that type of instance it cannot be considered that the user is a malicious user. So, in such situations, the trust analysis method is used for detecting whether that user is a malicious user or not.

Trust evaluation. In this evaluation, the user gives ratings during the specific time interval exhibited as the malicious user. In the specific time interval, typical users also give the rating value. So, malicious users are identified by the trust evaluation method. Here a belief hypothesis is utilized that is assigning a trust value to every user. The trust value relies upon their good and bad behavior. Having one limiting value, if the trust value falls below the limit value, then a user is considered as a malicious user. A trust value over the limit value means that a user is considered to be a good user.

- Number of good ratings = without malicious users
- Number of bad ratings = with malicious users

Calculation of correlation. After identifying the malicious user, the user correlation can be calculated. The correlation is computed by the equation

$$\text{Suspicious score} = \frac{(\text{Probability of good ratings} + \text{Probability of bad ratings})}{(\text{Probability of total number of ratings})}$$

The suspicious score calculated for the specific item is then based on this method. After identifying the score, it is then exhibited that the item is a good item or bad item.

Identification of malicious user. Considering the time interval and rating provided with the same e-mail id the malicious user is then identified. In this event, there is a time variation and if that variation is large enough then that user is determined to be a Malicious User1. If the variation is comparatively small then that user is Malicious User2. If the variation is extremely small then that user is considered as Malicious User3. The user giving the rating with the same e-mail id and then taking into account the time variation allows for the malicious user to be identified.

Product reputation score. After identifying the malicious user for the specific item, the original rating for the item is calculated. After this, the item is identified as to whether it is good or bad. First, the item ratings are given and then they are arranged in ascending order based on the time the ratings are given. An attack can be detected by the time interval. That is, if the rating is provided within a specific time interval, then it is possible that a malicious attacker made it. If there is no attack that implies the item is good. If an attack is detected, then the following procedure of trust evaluation is performed. Here, a trust value is assigned

to every user. In the event that the user trust value is high, it implies that the user is good and there is no attack occurring. If the trusts value is low, it means there may be an attack taking place. In the step correlation, the suspicious score for the specific item is calculated and also the item is evaluated. The malicious user is identified based on the provided trust value. There are three sorts of malicious users available. If the variation in the time interval is large, then the Malicious User1 is considered. If the variation is comparatively small, then Malicious User2 is considered. And, if the variation in the time interval is very small, then Malicious User3 is considered. The users who are giving the right rating value are considered to be a good user. The right score of the item is identified and after that the item capability is demonstrated.

Tools and Techniques

There are a few modules that are used to execute this study.

System model. User demonstrates the feedback-based reputation systems as the system in which users give ratings to items. This model can depict numerous functional systems. For instance, purchasers give ratings to items on Amazon.com and reader's rate social news on Reddit.com. The items in the above systems are products and social news, respectively. It is considered that each user will give a rating to one thing at most once, and the rating values are integer values going from 1 to 5. In practice, reputation systems regularly permit users to give surveys as well. These surveys can likewise be untruthful. In this paper, the identification of dishonest ratings is considered. The analysis of untruthful surveys is beyond the extent of this paper, though the dishonest rating detection and untruthful review detection complement one another.

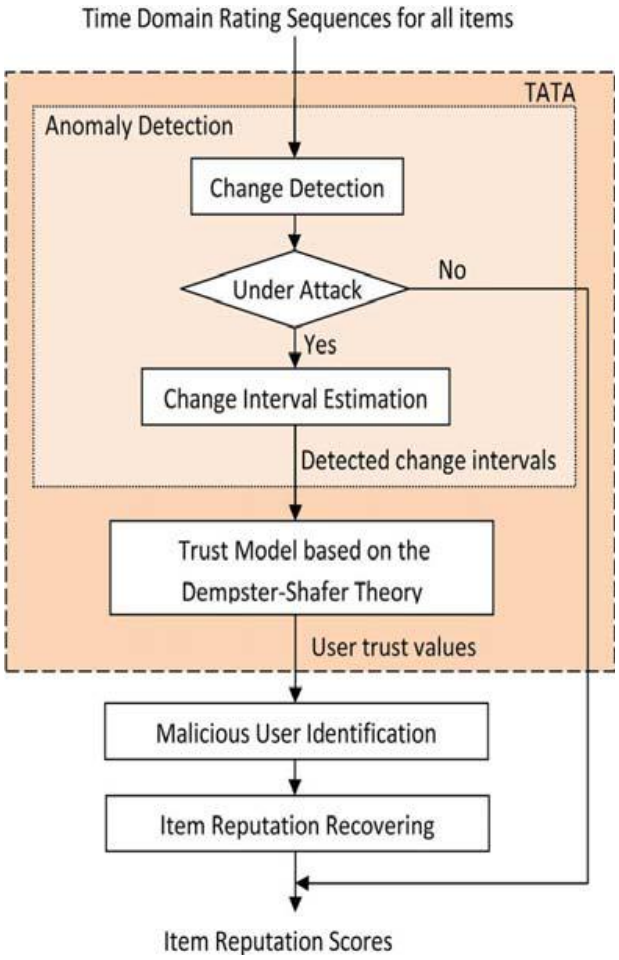


Figure 1. System Architecture

Attack model. An attacker can control one or more user's IDs and each of these user IDs is referred to as a malicious user. Malicious users give ratings to manipulate the reputation score of items. The item whose reputation score is manipulated by malicious users is known as a target item. The ratings given by malicious users to target items are considered to be dishonest ratings. An attack profile describes the behavior of all malicious users controlled by the attacker.

Assumption. In this work, it is assumed that items have characteristic qualities which do not change quickly. The rating values to a given item rely on the user's personal preference as well as the item quality. In a few applications, for example ratings for movies

or books, the item quality judgment is extremely subjective. In these examples, a users' personal preference assumes a more critical part of the equation. Though, in some different applications, for example Amazon item ratings, the item quality assumes an essential part. In this study, the product-rating type applications are focused on, especially where the rating distribution of an item is moderately stable. Hence, if rapid changes in the rating distribution happen it is possible that an anomaly has occurred.

Joint trust and temporal analysis. This is proposed to detect anomalies from a new angle that is analyzing time domain information. In particular, users organize the ratings to a given item as a sequence in the descending order as indicated by the time when they are given. This sequence, indicated by the timestamp, really reflects the rating pattern to a given item. Practically speaking, numerous items have similar characteristics and similar quality, which should be reflected in the distribution of ordinary ratings. If there are quick changes in the rating values, such changes can serve as pointers of an anomaly.

Change director. Numerous change detectors have been developed for diverse application situations. In online reputation systems, since typical ratings don't necessarily follow a particular distribution and attackers may embed dishonest ratings with little bias. There is a need to choose a change detector that is insensitive to the probability distribution of information and has the capacity to reliably recognize small bias in ratings. Hence, the CUSUM detector has been chosen, which satisfies these requirements as the base to build the change detector.

Basic and revised CUSUM. A basic CUSUM detector is introduced, which determines if a parameter in a probability density function (PDF) has changed. Then, a modified CUSUM detector is developed to estimate the change starting time and ending time. The modified CUSUM can detect numerous change intervals.

For a particular change interval, it is signified by the beginning time of the change and the closing time of the change.

Hardware and Software Environment

Hardware configuration:

Processor	Intel Core i7
RAM	4GB
Hard Disk	424GB

Software Configuration:

Operating System	Windows 8
Programming Language	JAVA
Java Version	JDK 1.6 and above.
Frontend	JSP, servlets
Backend	Oracle10g

Chapter IV: Implementation

Modules

1. Online Shopping Module
2. User Rating Module
3. Data Collection Module
4. Change Detection
5. Identify and Block Malicious Users

Modules Description

Online shopping module. In this module, a site is developed for web-based shopping. The client can buy items and then has the option to give appraisals and their proposals as criticism. In this module, the administrator can include product details (item name, value, legitimacy and so on) based on the classification of device like mobiles, PCs, laptops and so on while maintaining details of a product. The client enters their payment card details for validation and awaits approval. If the card details are legitimate, the client can then buy their items. The client can choose products that are to be purchased, which are then shown on the home page, or search for the item utilizing keywords or item classifications. The client can then buy the item utilizing an authorized payment card. To finalize the purchase the client is required to enter the details such as the card number, cardholder name, date of birth, and credit card issuer. If the card is found to be legitimate then the client is permitted to buy the item.

User rating module. In this module, the client is permitted to have the ability of providing their input in a type of ratings with respect to the service provider. Client ratings are considered as one of the essential variables in the purchasing decision as they assume a key role in whether an item is purchased or not. Misleading ratings may create extreme

problems in numerous feedback rating systems. Thus in this module, the client ratings are gathered and are secured so as not to allow tampering with them.

Data collection module. In this module, all client profiles and ratings are gathered. Client profile values are additionally integrated with their system IP address, timespan spent on the system and rating values provided. All of the client profiles including the values of the ratings are saved safely for later analysis.

Change detection module. In this module, the information gathered is utilized as a dataset. In the dataset the fraudulent clients are distinguished by their client name by sudden detection of changes. The chart (What chart are you referring to here?) shows the client fraud rate crosswise over months and measures the weight for tests within the client dataset.

Identify and block malicious users. In this module, the system is developed with the end goal that an administrator of the feedback rating system can have the ability to block the malicious clients. In this way the malicious clients cannot provide unfair or misleading feedback on the system.

Chapter V: Data Presentation and Analysis

Introduction

Provide a brief introduction on what will be covered in this chapter or in other words what should a reader expect to read from this chapter.

Data Presentation

In this section, the system design is explained in a detailed manner.

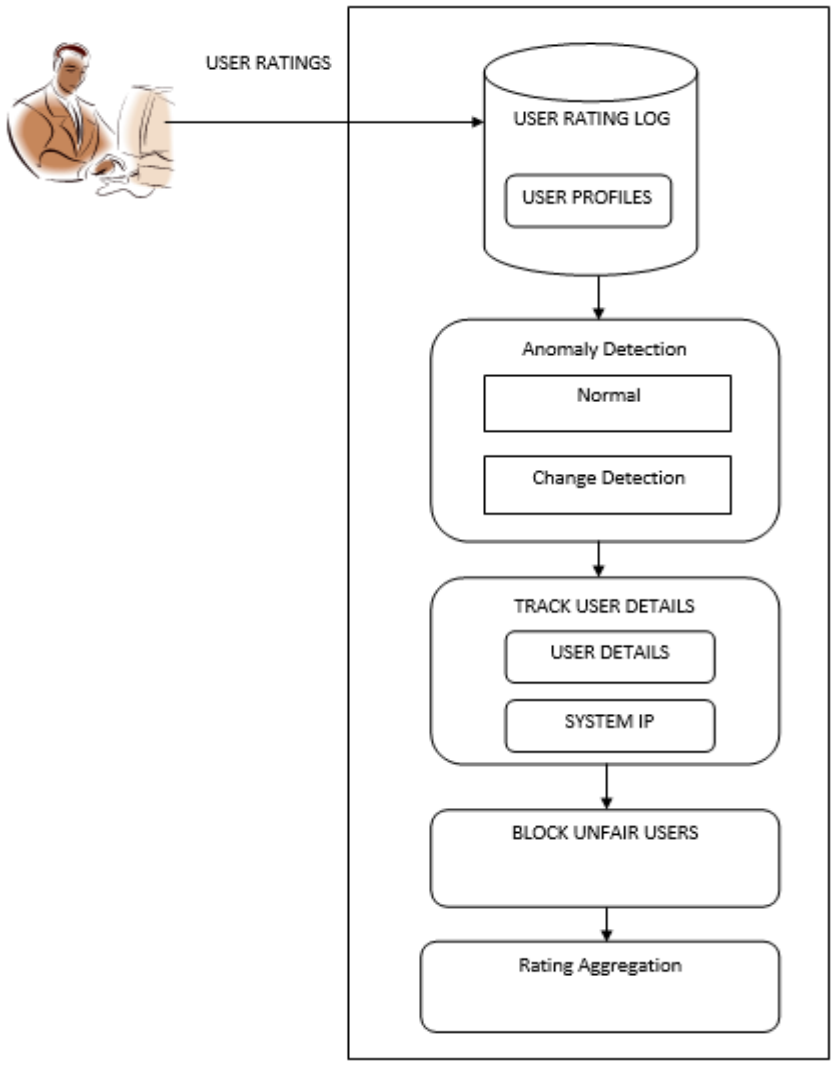


Figure 2. System Design

Data flow diagram.

- The Data Flow Diagram is also known as a bubble outline. It is a straightforward graphical illustration that can be utilized to represent a system regarding input information to that system. The data is handled differently including the output data which is created by the system.
- The data flow diagram (DFD) is a standout among the most critical modeling tools. This can be used to model the system components. These components are the various processes of the system, the data used by the process, and an external entity that associates with the system and the data streams in the system.
- DFD shows how the data travels through the system and how it is altered by a series of changes. It is a graphical illustration that portrays data streams and the changes that are applied as information moves from the source to target system.
- DFD can be utilized to represent a system at any level of deliberation. DFD can be partitioned into levels that represent expanding data streams and functional details.

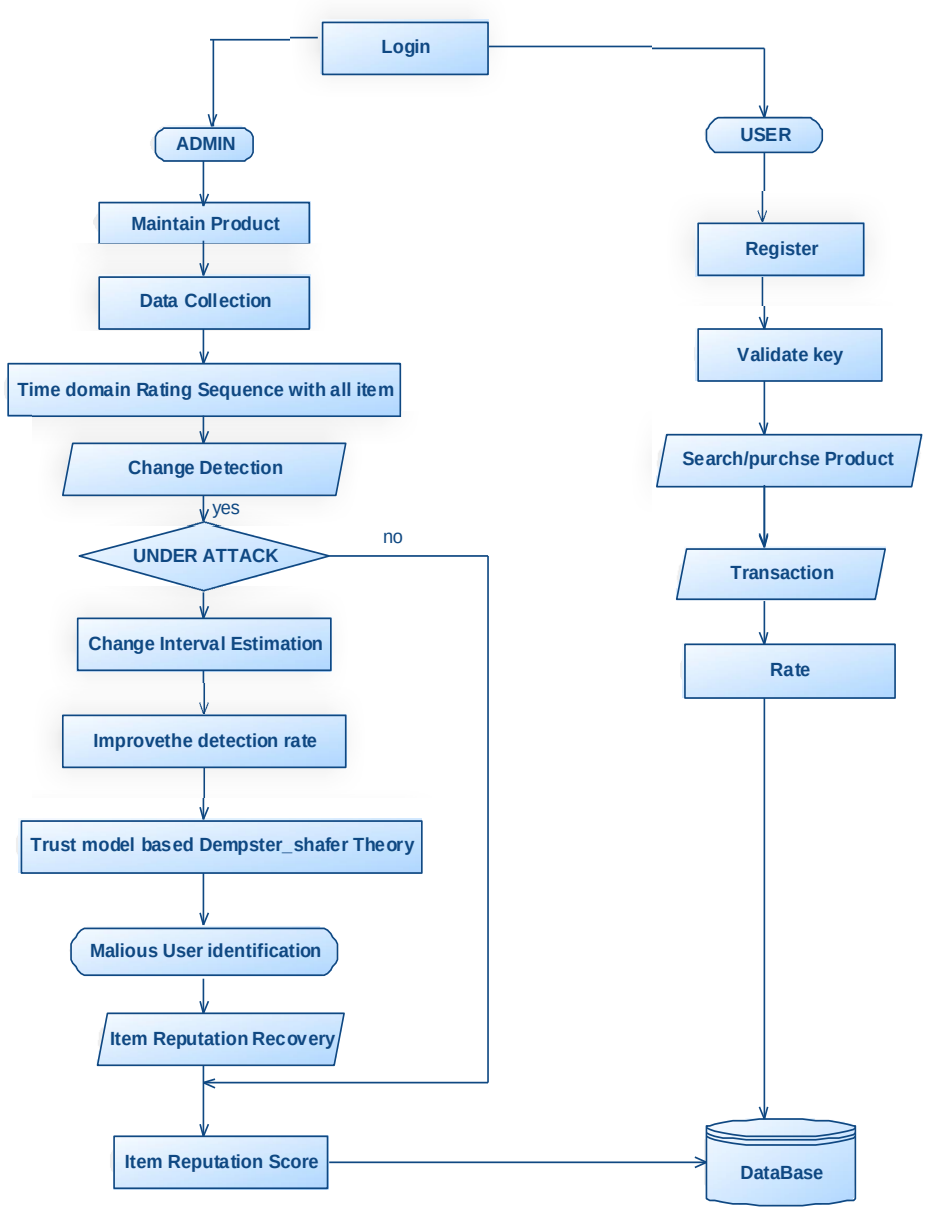


Figure 3. Data Flow Diagram

UML diagrams. Unified Modeling Language (UML) is considered to be a broadly used modeling language in the field of object-oriented software engineering. The Object Management Group were the ones who created and have managed UML. The objective of UML is to have a common language for creating models of object-oriented computer software. In its present state, UML comprises two important components, a meta-model and a

documentation framework. Later on, some type of strategy or process may also be added to or connected with the UML.

The UML is a standard language for indicating, visualization, constructing and reporting the artifacts of a programming system. And, in addition, it is also used to provide business modeling and other non-programming systems. The UML represents an accumulation of best engineering practices that have had demonstrated success in the modeling of expansive and complex systems. UML plays a critical role in creating object-oriented software and the software development process. Generally speaking, UML utilizes graphical notations to express the plan and outline of programming projects.

Goals. The primary objectives in the design of the UML are:

- Provide clients with a ready-made, expressive visual modeling language with the goal that they can create and trade important models.
- Provide extensible and specialization systems to augment the core concepts.
- Being independent of specific programming languages and development processes.
- Provide a formal reason for understanding the modeling language.
- Encourage the development of object-oriented devices to market.
- Supporting higher level development concepts including systems, structures, collaborations, and segments. Integrate best practices and procedures.

Use case diagrams. A use case diagram in the Unified Modeling Language (UML) is a kind of behavioral outline characterized by and created from a use case analysis. Its purpose is to introduce a graphical outline of the functionality given by a system in terms of actors, their objectives and any conditions between those use cases. The primary reason for a use

case outline is to show the functions of a system that are performed for which actor. Roles of the characters in the system can be portrayed graphically.

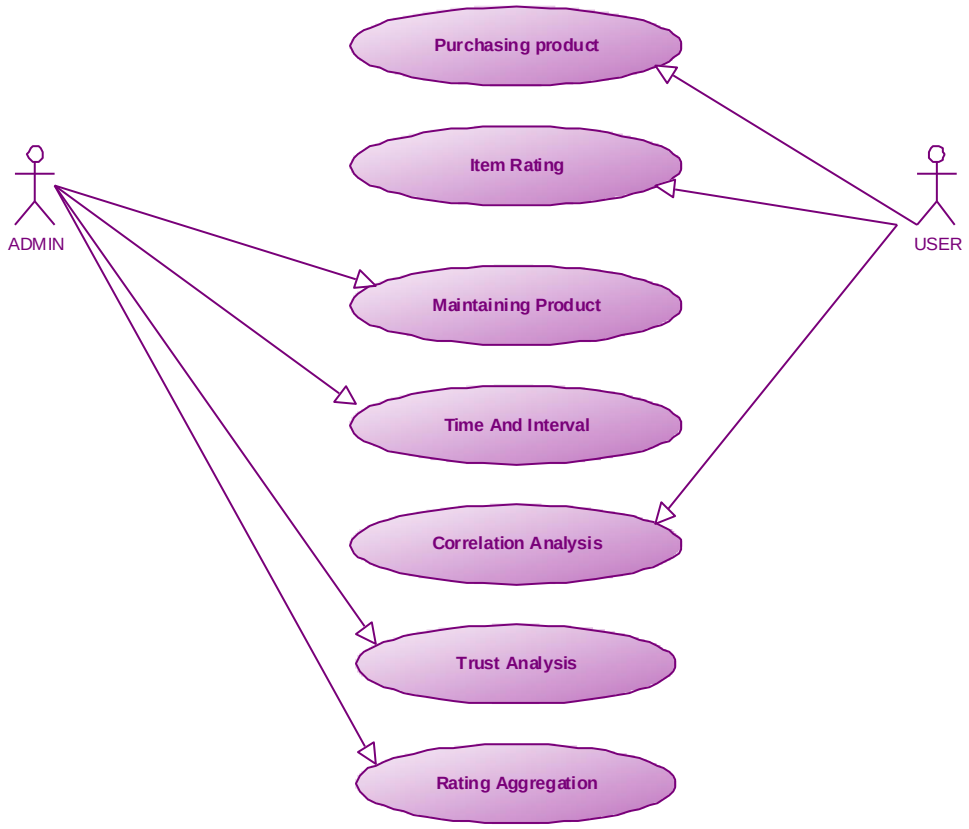


Figure 4. Use Case Diagram

Class diagram. A class diagram in the Unified Modeling Language (UML) is a kind of static structure chart that depicts the structure of a system by demonstrating the system's classes, their traits, operations (or strategies), and the relationships among the classes in software engineering. Also, a class diagram explains which class contains which data.

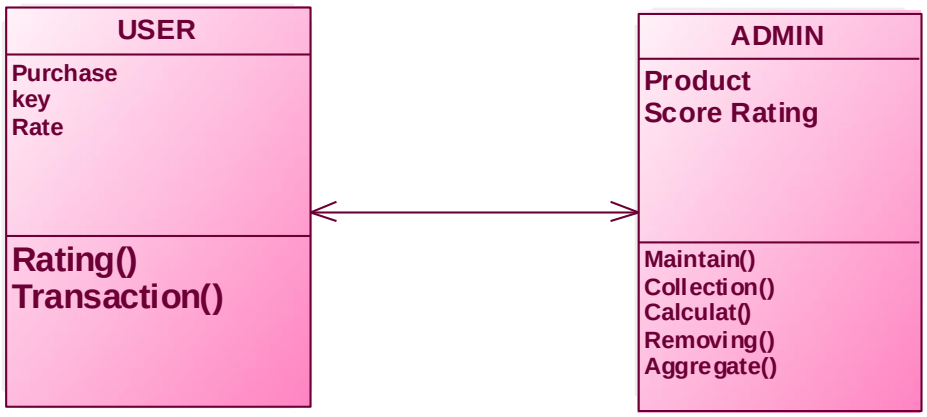


Figure 5. Class Diagram

Sequence diagram. In Unified Modeling Language (UML), the sequence diagram is a kind of interaction diagram that shows how processes work with each other in an orderly manner. It is built from a message sequence chart. The message sequence chart diagrams are also known as event graphs, event situations, and timing charts.

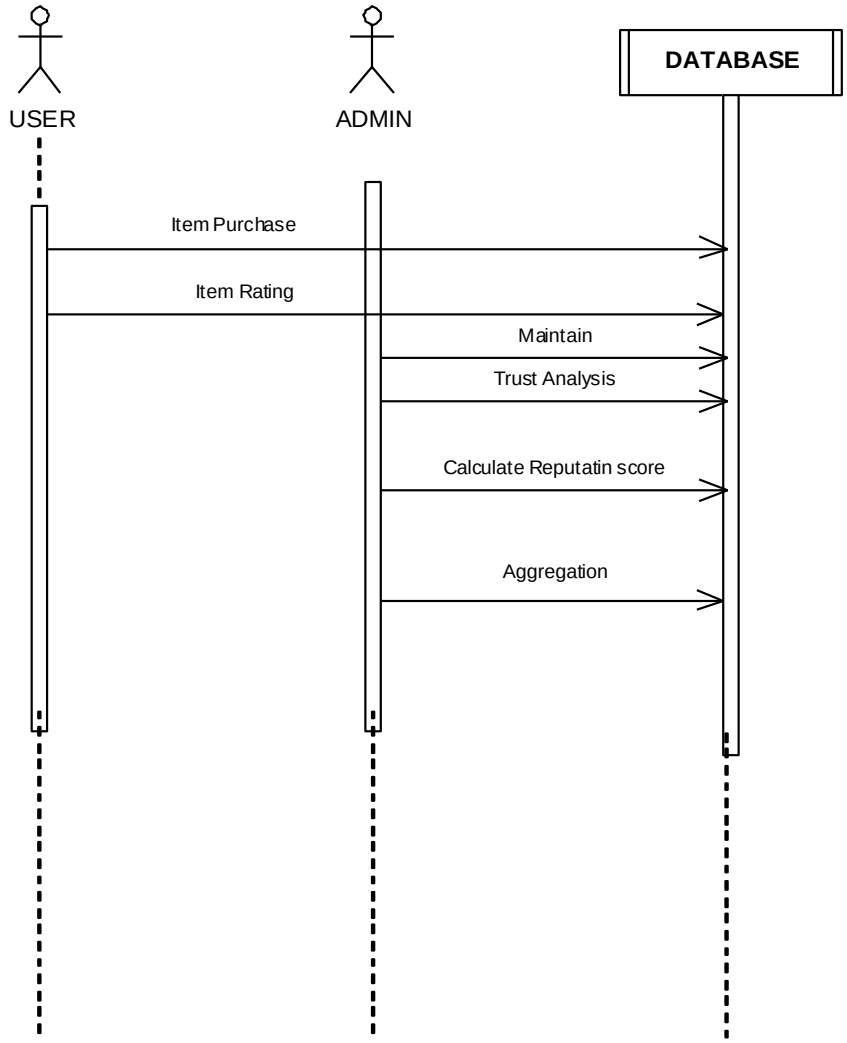


Figure 6. Sequence Diagram

Activity diagram. Activity diagrams are graphical portrayals of work processes of tasks and activities. They provide support for the decision, cycle, and concurrency of these tasks. In the Unified Modeling Language, activity diagrams can be utilized to portray the business and operational well-ordered work processes of components in a system. The overall flow of control within a system is demonstrated by the activity diagram.

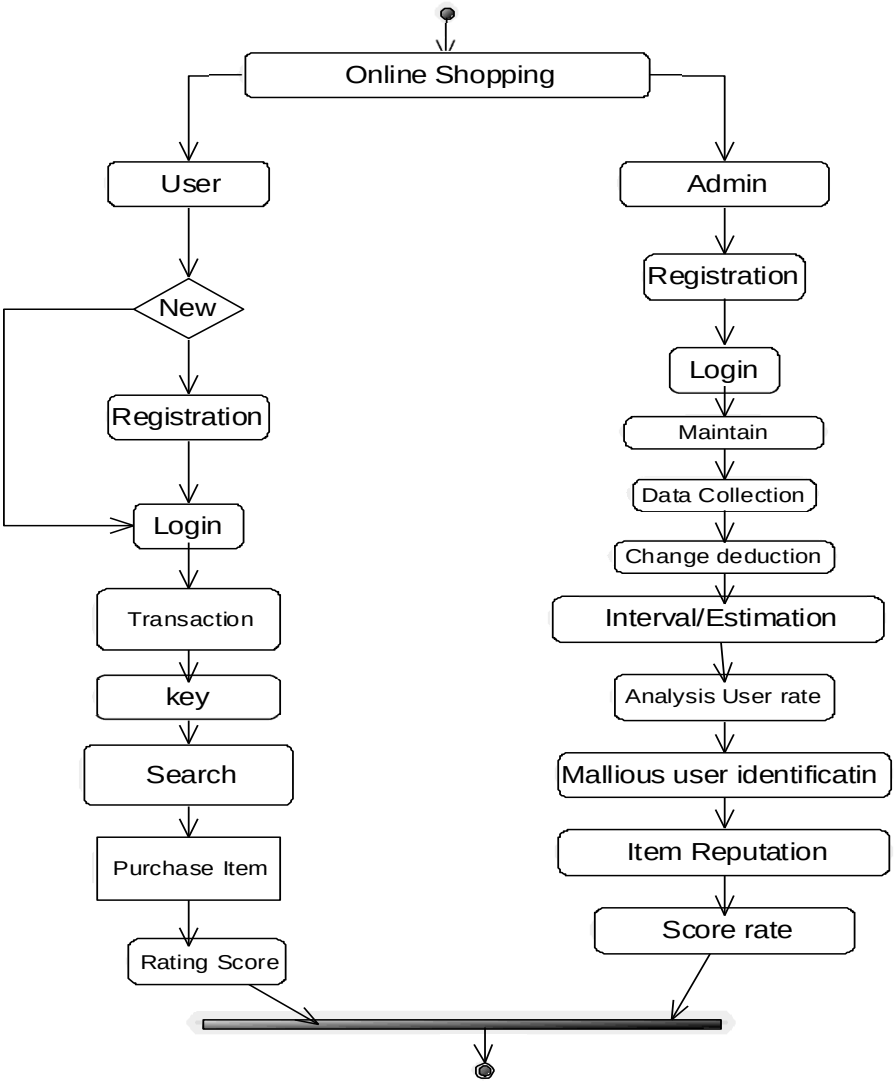


Figure 7. Activity Diagram

Data Analysis

Feasibility study. The feasibility of the project is investigated in this stage and the business proposition is advanced with a general plan for the project and some cost estimates. The feasibility study of the proposed work during the system analysis stage would need to be done. This is to guarantee that the proposed system is not a financial burden to the organization. For feasibility analysis, some understanding of the significant necessities for the system is required.

Three key considerations are involved in the feasibility analysis they are: Economic Feasibility, Technical Feasibility, and Social Feasibility.

Economic feasibility. This study is done to check the financial effect that the system will have on the organization. The funds that the organization can afford to create the innovative work of the system is restricted by budgetary constraints. There is also a need to justify the expenditures. In this manner, the developed system will need to fit within the financial plan of the organization. This was accomplished in large part by the fact that a significant portion of the technologies utilized are open source and are freely accessible. Only the customized items for the feedback system must be purchased.

Technical feasibility. This study is done to check the technical feasibility of the system, including the technical requirements of implementing it. Any system developed must not be overly burdensome on the available technical assets. This will prompt levels of popularity on the accessible technical assets of the organization as well as on the customer. There must be reasonable requirements for the developed system, so as negligible or invalid changes are not required for executing this system.

Social feasibility. This study is to check the level of acknowledgment of the system by the client. This incorporates the way toward preparing the client to utilize the system effectively. The client must not feel debilitated by the system, but should rather acknowledge it as a need. The level of acknowledgment by the clients exclusively relies on upon the techniques that are utilized to instruct the client about the system and to make them acquainted with it. Their level of certainty in use of the system must be raised with the goal that they are ready to create some productive feedback around an item or service, which then can be used by other clients of the system.

The motivation behind testing systems is to find any errors that may occur. Testing is a way toward attempting to find possible errors or shortcomings in a product or service. It provides an approach to check the usefulness of the components, sub gatherings, assemblies and eventually the completed product. It is a way of practicing software development with the aim of guaranteeing that the software system lives up to its requirements and meets the clients' desires and does not fail in an unsatisfactory manner. There are different types of tests that can be performed. Each type of test sorts addresses and covers a particular testing requirement.

Types of Tests

Unit testing. Unit testing includes the outline of test cases that assure that the inside programmatic logic is working appropriately, and that program inputs create substantial and correct outputs. All choice branches and flow of internal code need to be approved. It is necessary to test all of the individual software units of the application or system. This is done after an individual unit of the software is finished but before coordination of the components in the system. This is considered a basic form of testing, and depends on learning of its development and is obtrusive. Unit tests perform essential tests at the segment level and test a particular business process, application, and system design. Unit tests guarantee that each different way of a business procedure performs is precisely functioning according to the archived details and contains plainly characterized inputs and expected outcomes.

Unit testing is normally led as a major aspect of a combined code and unit test period of the software life cycle, this is done in spite of the fact that it is normal for coding and unit testing to be directed at two distinct stages.

Test strategy and approach. Field testing will also be performed physically, and functional tests will be composed in detail.

Test objectives.

- All field entries must work legitimately
- Pages must be activated from the distinguished connection
- The entry screen, messages, and reactions must not be deferred

Features to be tested.

- Verify that the entries are in the right configuration
- No copy entries should be permitted
- All connections should take the client to the appropriate page

Integration testing. Integration tests are intended to test the incorporated components of software to figure out whether they really keep running as one program. Testing is event-driven and worries more about the essential results of the screens. Integration tests are used despite the fact that the components were tested exclusively from one another and appeared to be effective in unit testing. The integration test assures the blend of segments is operating correctly and is stable. Integration testing is particularly aimed at uncovering the issues that emerge from the blending of components.

Functional testing. Functional tests give systematic demonstrations of previous tests of the functions of the system that are accessible as determined by the business and technical necessities, system documentation, and client manuals.

Functional testing is centered on the following items:

- *Valid Input*: Distinguished classes of legitimate inputs must be acknowledged.
- *Invalid Input*: Distinguished classes of invalid inputs must be rejected.
- *Functions*: Distinguished functions must be implemented.
- *Output*: Distinguished classes of utilization outputs must be implemented.
- *Systems/Procedures*: Interfacing systems or techniques must be queried.

Organization and planning of functional tests are centered on prerequisites, key functions, or special test cases. Moreover, the systematic scope as it relates to recognizing business process flows, information fields, predefined forms, and progressive procedures must be considered for testing. Before functional testing is completed, more tests are recognized, and the viable estimation of current tests is resolved.

Software integration testing is the incremental coordinated testing of at least two incorporated software segments on a solitary stage to deliver failures which are caused by defects of the interface. The task of the integration test is to determine whether segments or software applications (e.g. components in a software system or software applications at the organization level) communicate without errors or mistakes.

Test results. All the experiments specified above passed effectively and no imperfections were experienced.

System testing. System testing guarantees that the whole integrated software system meets the prerequisites that were created. It tests a design to guarantee known and predictable results. An example of system testing is the design arranged system reconciliation test. System testing depends on process descriptions and streams, underlining pre-driven process

White box testing. White Box Testing is a testing in which the software analyzer knows about the inward workings, structure and language of the software, or its motivation. It is utilized to test cases that cannot be reached from a black box level.

Black box testing. Black Box testing is testing the software with no information of the internal workings, structure or language of the module being tested. Black Box tests must be composed of a conclusive source archive, for example, a determination or necessities document or a particular prerequisites report. It is a form of testing in which the software

under test is dealt with as a black box that is unknown. The test gives data sources and reacts to outcomes without considering how the software functions.

Acceptance testing. User Acceptance Testing is a crucial period of any project and requires significant investment by the end user or client. It likewise guarantees that the system meets the functional prerequisites that were set for it.

Test results. All the experiments specified above passed effectively and no imperfections were experienced.

Screenshots

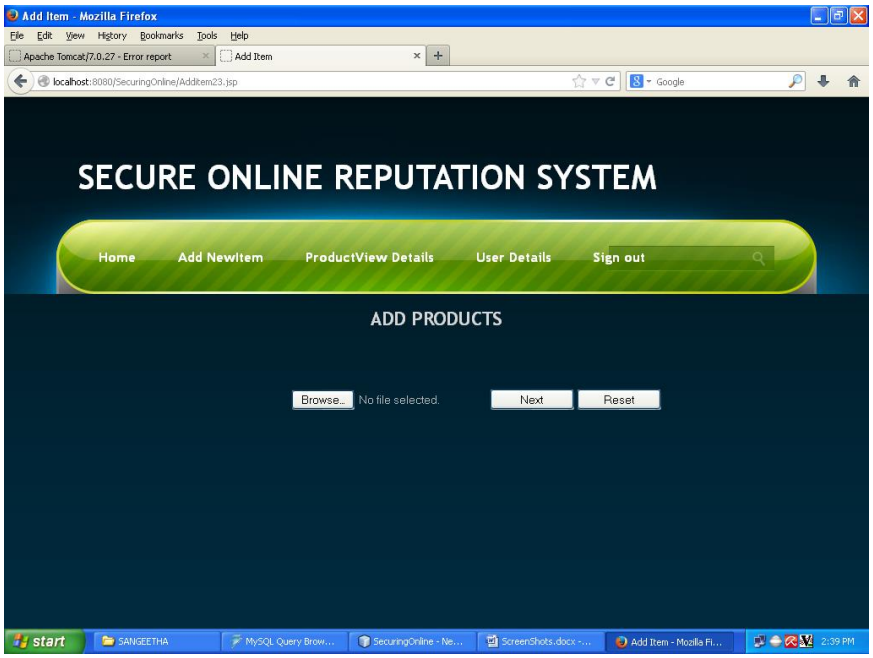
Step 1: Welcome Page. The User can login to the website through this page and access the dashboard.

Step 2: User Login. This page gives access to the user to enter their credentials to log in to a particular account for online shopping.

Step 3: Search for Products. Once the user logs into the website, and through accessing the dashboard they can start their online shopping by clicking on the category "search by products."

Step 4: Product Gallery. This page gives all the details of a product including product name, product item, brand name, price, and validity.

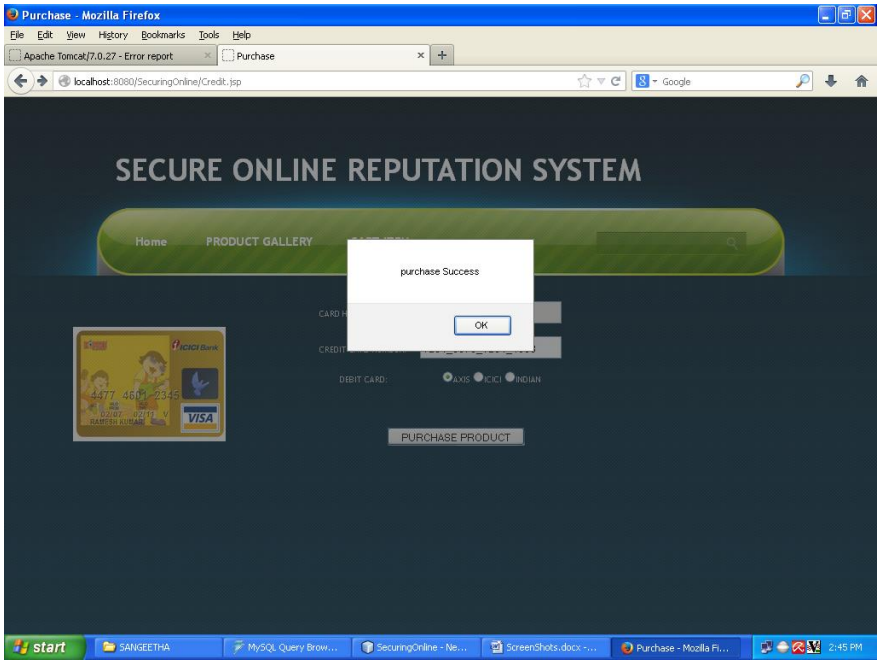
Step 5: Adding New Item. Through this page, the user can add a new product to their cart and continue their shopping.



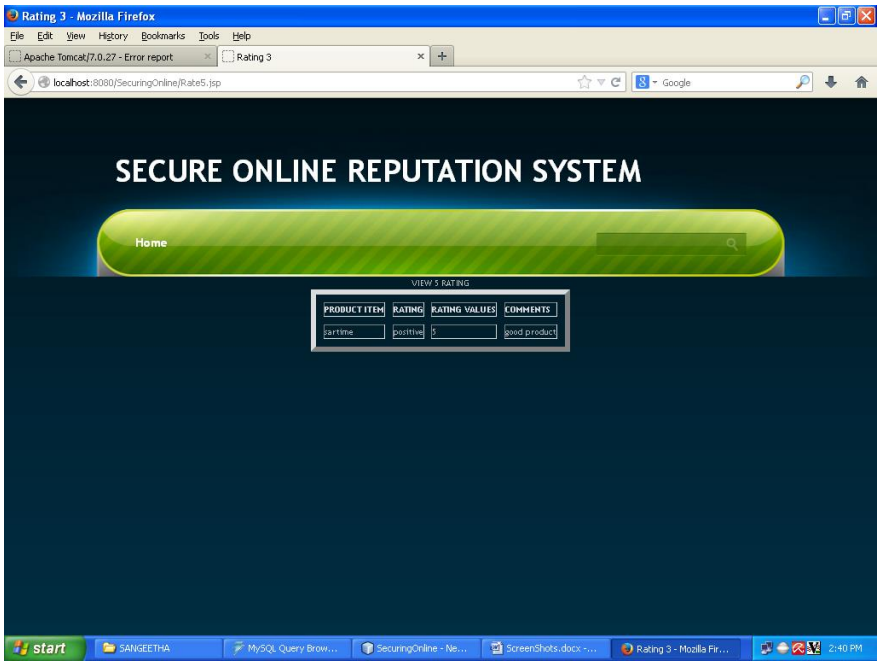
Step 6: Purchased Products. This page gives the details of all the products that have been purchased.

Step 7: Card Holder. This page allows the user to enter their payment card details in order to purchase products.

Step 8: Purchase Success. This page gives the confirmation of a successful purchase of the product.

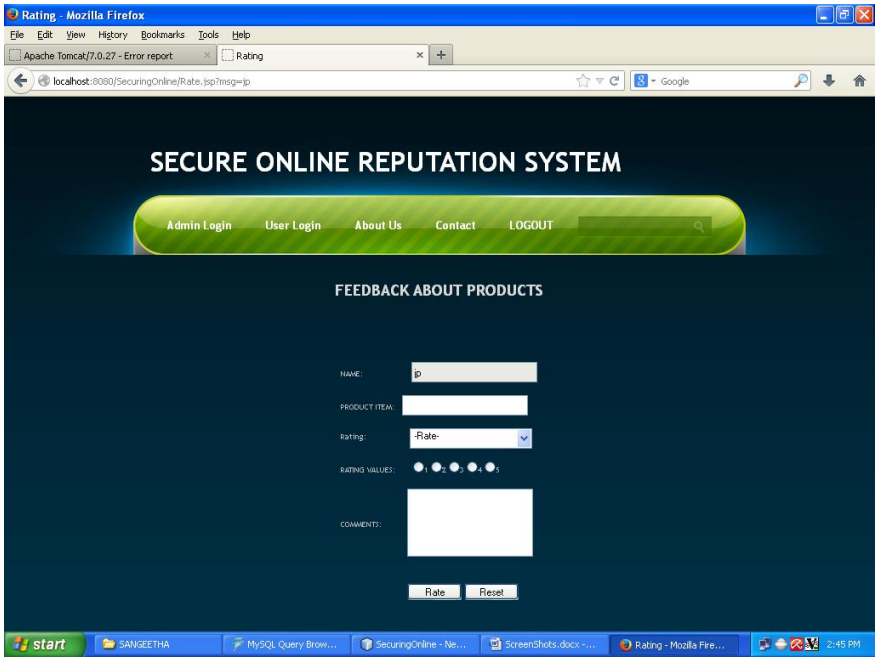


Step 9: Rating Page. This page shows the rating values given by the users.



Step 10: Feedback. In this page, all of the users' feedback ratings are secured and can be viewed by the administrator for product review analysis.

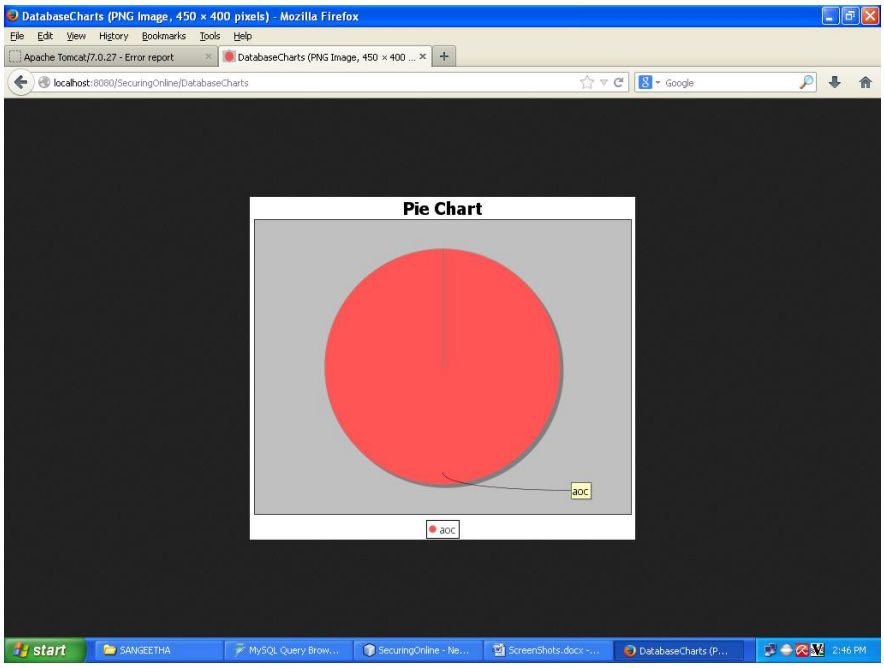
Step 11: Feedback About Products. This page allows the users to provide feedback about the products they have purchased.



Step 12: Admin Login. This page gives access to the administrator to login and check details on different user's accounts.

Step 13: User Details. This page lists all of the registered user details like user Id, username, DOB, email id, location and contact details.

Step 14: Rating Graph. This chart gives the analysis of all the ratings given by users.



Chapter VI: Conclusion

Many schemes have exhibited great performance in ensuring reputation feedback systems, however there are still problems that have not been completely addressed. In this paper, an extensive anomaly detection scheme (TATA) is designed and assessed for ensuring feedback of online reputation systems. Temporal and Trust analysis (TATA) exhibits critical advantages such as recognizing things under attack, distinguishing malicious clients who embed dishonest ratings and recovering reputation scores. To investigate the time-domain data, a modified CUSUM detectors were developed to identify change in the interim. Online reputation systems are progressively influencing individuals' online shopping and downloading decisions. Also, as evidenced by the project, it has been shown that the Temporal and Trust Analysis reputation system works and provides significant benefits over other systems.

References

Falcone, R., & Castelfranchi, C. (2001). *Social trust: A cognitive approach*. AA Dordrecht, The Netherlands: Kluwer.

Freeman, L.C. (1979). Centrality on social networks. *Social Networks, 1*, 215-239.

Gambetta, D. (1990). Can we trust? In D. Gambetta (Ed.),*Trust: Making and breaking cooperative relations* (pp. 213-238). Oxford: Basil Blackwell.

Liu, Y., & Sun, Y. (2010). Anomaly detection in feedback-based reputation systems through temporal and correlation analysis. In *Proceedings of the Second IEEE International Conference on Social Computing* (pp. 65-72).

Marsden, P.V., & Lin, N. (1982). *Social structure and network analysis*. Beverly Hills, CA: Sage Publications.

McKnight, D.H., & Chervany, N. L. (1996). *The meanings of trust*. Technical Report MISRC Working Paper Series 96-04: St. Paul, MN: University of Minnesota, Management Information Systems Research Center.

Reputation. (n.d.). *The Concise Oxford Dictionary* (8th ed.). Oxford: Oxford University Press.

Resnick, P., Zeckhauser, R. Friedman, R., & Kuwabara, K. (2000). Reputation systems. *Communications of the ACM, 43*(12), 45-48.

Whitby, A., Josang, J., & Indulska, J. (2005). Filtering out unfair ratings in Bayesian reputation systems. *Icfain Journal of Management Research, 4*(2), 48-64.

Yu, H., Kaminsky, M., Gibbons, P. B., & Flasman, A. (2006). Sybilguard: Defending against Sybil attacks via social networks. In *Proceedings of Conference Applications, Technologies, Architectures and Protocols for Computer Communication*, pp. 267-278).

Appendix

Additem:

```
import Utility.Dbconn;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.tomcat.util.http.fileupload.FileItem;
import org.apache.tomcat.util.http.fileupload.FileUploadException;
import org.apache.tomcat.util.http.fileupload.disk.DiskFileItemFactory;
import org.apache.tomcat.util.http.fileupload.servlet.ServletFileUpload;
```

```
public class Additem extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      response.setContentType(“text/html;charset=UTF-8”);
      PrintWriter out = response.getWriter();
      Connection con = null;
      try {
         File f = null;


         DiskFileItemFactory diskFile = new DiskFileItemFactory();
         diskFile.setSizeThreshold(1 * 1024 * 1024);
         diskFile.setRepository(f);
         ServletFileUpload sfu = new ServletFileUpload(diskFile);
         List item = sfu.parseRequest(request);
         Iterator itr = item.iterator();
         FileItem items = (FileItem) itr.next()
         Class.forName(“com.mysql.jdbc.Driver”);
         con = DriverManager.getConnection(“jdbc:mysql://localhost:3306/securing”, “root”,
“root”);
         System.out.println(“database connected”);
         //Statement st = con.createStatement();
         //   int i = st.executeUpdate(“insert into
additem(productname,productitem,brandname,itemcost,manaf_date,image) values(‘“ +
productname + “‘,’” + productitem + “‘,’” + brandname + “‘,’” + itemcost + “‘,’” +
manaf_date + “‘,?)”);
         PreparedStatement pstmt = null;
         pstmt = con.prepareStatement(“insert into additem(image,imgname) values(?,?)”);
```

```
            pstmt.setBinaryStream(1, items.getInputStream());
            pstmt.setString(2, items.getName());
  boolean i =   pstmt.execute();
System.out.println("Data is successfully inserted!  "+ i );
           response.sendRedirect("Additem23_1.jsp");

        } catch (ClassNotFoundException ex) {
            Logger.getLogger(Additem.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SQLException ex) {
            Logger.getLogger(Additem.class.getName()).log(Level.SEVERE, null, ex);
        } catch (FileUploadException ex) {
            Logger.getLogger(Additem.class.getName()).log(Level.SEVERE, null, ex);
        }  finally {
            out.close();
        }
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP
     * <code>POST</code> method.
     *
```

```
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return “Short description”;
    }// </editor-fold>
}
```

CreditCard:

```
/*
* To change this template, choose Tools | Templates
 * and open the template in the editor.
```

```
 */
package Action;

import Utility.Dbconn;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author m
 */
public class Credit extends HttpServlet {

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
```

```
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        Connection con = null;
        try {



            String username = request.getParameter("user");
            String card_no = request.getParameter("credit");
            Class.forName("com.mysql.jdbc.Driver");


            con = Dbconn.getConnection();


            Statement st = con.createStatement();
            ResultSet stud = st.executeQuery("select * from register where name= '" + username
+ "'");


            if (stud.next()) {


                if (card_no.equals(stud.getString("card_no"))) {
```

```
                System.out.println("login success");
                // session.setAttribute("UID", username);
                out.println("purchase success");
                response.sendRedirect("Rate.jsp?msg="+username);
                //response.sendRedirect("Product1.jsp?msg="+username);


            } else {
                out.println("credit card number error....!");
            }

        } else {

            out.println("Useraname error....!");
            System.out.println("Useraname error....!");
        }
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(Credit.class.getName()).log(Level.SEVERE, null, ex);
    } catch (SQLException ex) {
        Logger.getLogger(Credit.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        out.close();
    }
}
```

```
    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
    /**
     * Handles the HTTP
     * <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP
     * <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
```

```
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return “Short description”;
    }// </editor-fold>
}
```

RateEditing:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package Action;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 *
 * @author m
 */
public class EditRate extends HttpServlet {

    /**
     * Processes requests for both HTTP
```

```
* <code>GET</code> and
* <code>POST</code> methods.
*
* @param request servlet request
* @param response servlet response
* @throws ServletException if a servlet-specific error occurs
* @throws IOException if an I/O error occurs
*/
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {


        HttpSession session = request.getSession(true);
        String UserID = session.getAttribute("UID").toString();
        System.out.println("getUser ID IS :" + UserID);
        String productname = request.getParameter("productname");
        String productitem = request.getParameter("productitem");
        String brandname = request.getParameter("brandname");
        String itemcost = request.getParameter("itemcost");
        String itemqty = request.getParameter("qty");
```

```
        //System.out.println(name + "\n" + m1+ "\n" + m2+ "\n" + m3+ "\n"
+m4+"\n"+m5+"\n"+percen+"\n");

        //Connection con = Dbc.con();


        //con = Dbc.getConnection();//(Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/student", "root", "root");

        Class.forName("com.mysql.jdbc.Driver");

        int getcost = 0;

        int qty = Integer.parseInt(itemqty);

        int total = 0;

        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/securing", "root", "root");

        Statement st = con.createStatement();

        Statement st1 = con.createStatement();

        ResultSet rs = st1.executeQuery("select * from additem where productitem = '" +
productitem + "'  ");

        if (rs.next()) {

            getcost = rs.getInt("itemcost");

        }

        total = qty * getcost;


        System.out.println("total is " + total);


        //   int i = st.executeUpdate("update additem set productitem='" + productitem +
"',brandname='" + brandname + "',itemcost='" + itemcost + "' where name='" +
productname + "'  ");

        int i = st.executeUpdate("insert into card values('" + UserID + "','" + productname +
"','" + productitem + "','" + itemqty + "','" + total + "')");
```

```
            // out.println("Data is successfully Updated!");
             response.sendRedirect("Product1.jsp?");
             // response.sendRedirect("Credit.jsp?");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(EditRate.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SQLException ex) {
            Logger.getLogger(EditRate.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            out.close();
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
    /**
     * Handles the HTTP
     * <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
```

```
    processRequest(request, response);
}

/**
 * Handles the HTTP
 * <code>POST</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Returns a short description of the servlet.
 *
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
```

```
        return “Short description”;
    }// </editor-fold>
}
```

Updating Items:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package Action;


import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;


/**
```

```
 *
 * @author m
 */
public class Update extends HttpServlet {

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {



            //  HttpSession session = request.getSession(true);
            String productname = request.getParameter("productname");
```

```
        System.out.println(productname);
        String productitem = request.getParameter("productitem");
        String brandname = request.getParameter("brandname");
        String itemcost = request.getParameter("itemcost");
        String manaf_date = request.getParameter("manaf_date");




        Class.forName("com.mysql.jdbc.Driver");


        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/securing", "root", "root");
        Statement st = con.createStatement();
        System.out.println("database connected");


        int i = st.executeUpdate("update additem set itemcost='" + itemcost + "' where
productitem='" + productitem + "'  ");
        response.sendRedirect("ProductView.jsp?");



      } catch (ClassNotFoundException ex) {
        Logger.getLogger(Update.class.getName()).log(Level.SEVERE, null, ex);
      } catch (SQLException ex) {
        Logger.getLogger(Update.class.getName()).log(Level.SEVERE, null, ex);
      } finally {
```

```
            out.close();
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
    /**
     * Handles the HTTP
     * <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP
     * <code>POST</code> method.
     *
     * @param request servlet request
```

```
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return “Short description”;
    }// </editor-fold>
}
```

Register:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package Action;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author m
 */
public class RegisterA extends HttpServlet {

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
```

```
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        Connection con = null;
        try {

            String productname = request.getParameter("product");
            String productitem = request.getParameter("item");
            String price = request.getParameter("price");
            String validity = request.getParameter("validity");
            String brandname = request.getParameter("name");
            String description = request.getParameter("description");
            // String validity = request.getParameter("validity");
            //  String web_id = request.getParameter("mail");

            //String country = request.getParameter("country");


            System.out.println(productname + "\n" +productitem+"\n"+ price + "\n" + validity +
"\n" + brandname + "\n" + description + "\n");
```

```
            //Connection con = Dbc.con();


            //con = Dbc.getConnection();//(Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/student", "root", "root");

            Class.forName("com.mysql.jdbc.Driver");

            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/securing", "root",
"root");

            System.out.println("database connected");

            Statement st = con.createStatement();

            int i = st.executeUpdate("insert into
item(productname,productitem,price,validity,brandname,description) values('" +
productname + "','" + productitem + "','" + price + "','" + validity + "','" + brandname +
"','" + description + "')");

            out.println("Data is successfully inserted!");


        } catch (ClassNotFoundException ex) {

            Logger.getLogger(RegisterA.class.getName()).log(Level.SEVERE, null, ex);

        } catch (SQLException ex) {

            Logger.getLogger(RegisterA.class.getName()).log(Level.SEVERE, null, ex);

        } finally {

            out.close();

        }

    }


    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on
the left to edit the code.">

    /**

     * Handles the HTTP
```

```
 * <code>GET</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Handles the HTTP
 * <code>POST</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
```

```
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return “Short description”;
    }// </editor-fold>
}
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

    JDBCPieDataset dataset = new JDBCPieDataset(con);
    try {
      dataset.executeQuery(“Select `productitem`, `itemcost` From additem order by itemcost desc”);
      JFreeChart chart = ChartFactory.createPieChart(“Pie Chart”, dataset, true, true, false);
      if (chart != null) {
        response.setContentType(“image/png”);
        OutputStream out = response.getOutputStream();
```

```
        ChartUtilities.writeChartAsPNG(out, chart, 450, 400);
      }
    }
    catch (SQLException e) {
      e.printStackTrace();
    }
    try {
      if(con != null){con.close();}
    }
    catch (SQLException e) {}
  }

}
```